pysvn - Programmer's reference

This programmer's reference gives complete and detailed infomation on the pysvn API.

The pysvn Programmer's Guide gives an tutorial introduction to the pysvn module.

pysvn features and feature testing

This document covers pysvn version 1.7. Features offered by pysvn depend on the version of SVN that is being used. Full functionality i only available with SVN 1.6.0 or later.

Click one of the buttons below to show the pysvn API as supported by a particular version of the SVN.

Show SVN 1.9.0 API	Show SVN 1.8.0 API	Show SVN 1.7.0 API	Show SVN 1.6.0 API
Show SVN 1.5.0 API	Show SVN 1.4.0 API	Show SVN 1.3.0 API	Show SVN 1.2.0 API
Show SVN 1.1.0 API			

(•) Hilite unsupported ( ) Hide unsupported

Showing the PySVN API supported by SVN 1.9.0. Unsupported parts of the API are show like: This.

The recommended way to test for a feature is to use the python hasattr() builtin. Working out what is and is not support from the versi number information is quite complex and unnessesary. For example to test for lock and unlock support:

```
client = pysvn.Client()
if hasattr( client, 'lock' ):
    # use lock
```

pysvn module

The pysvn module has the following variables:

- pysvn.copyright - the pysvn copyright string
- pysvn.version - the pysvn version as a tuple, (major, minor, patch, build)
- pysvn.svn_version - subversion version as a tuple (major, minor, micro, tag)

The pysvn module has six classes:

- Client - the subversion client interface
- Transaction - the subversion transaction interface
- Revision - subversion revision objects
- ClientError - Exception class raised by client commands on error
- PysvnStatus - subversion status object
- PysvnEntry - subversion entry object

The following enumerations are provided:

- opt_revision_kind - kinds of Revision object
- wc_notify_action - see Client.callback_notify
- wc_status_kind - see Client.status()
- wc_schedule - see Client.status()
- wc_merge_outcome - see Client.Merge()
- wc_notify_state - see Client.callback_notify
- node_kind - see Client.status() and Client.ls()
- depth - replacement for recurse

Use python builtin dir() to list all available values in an enumeration:

```
    print dir( pysvn.wc_notify_action )
```

pysvn.Client - Subversion client interface

Interface summary:

```
client = pysvn.Client()
client = pysvn.Client( config_dir )
```

The default subversion configuration directory is used if the config_dir is omitted or set to ''.

The configuration directory is automatically created if it is missing.

A Client object can only be used on one thread at a time. If two threads attempt to call methods of Client at the same time one of the threads will get a pysvn.ClientError exception with the value 'client in use on another thread'.

Variables Callbacks Methods

Client variables

exception_style allows you to control the style of exception raised by pysvn.

commit_info_style allows you to control the style of commit_info returned by pysvn.

pysvn.Client.exception_style

exception_style is used to control how pysvn raises ClientError exceptions.

The default value, 0, makes pysvn raise exceptions as it did prior to pysvn 1.1.2.

exception_style can be set to 0 or 1, see ClientError for details of effect of the style on the exception raised.

pysvn.Client.commit_info_style

commit_info_style is used to control how pysvn return commit information.

commit_info_style can be set to 0, 1 or 2. The default value, 0, makes pysvn return only the commit revision.

When set to 1 pysvn returns a dictionary of commit information including date, author, revision and post_commit_err.

When set to 2 pysvn returns a list of dictionaries of commit information including date, author, revision and post_commit_err.

Client callbacks

pysvn uses callback functions to allow for realtime feedback and credential handling.

callback_cancel allows you to cancel a long running subversion command.

callback_notify gives feedback as commands runs.

callback_get_log_message is called when a log message is required.

callback_get_login is called to get a username and password to access a repository.

callback_ssl_server_trust_prompt is called when using HTTPS to a server whoes certificate needs is trust verifing.

callback_conflict_resolver is called to handle conflicts.

It is possible to use the Client object without setting up any calls backs. Make sure that all nessesary usernames, passwords and SSL certificate information are stored in the subversion configuration directory.

pysvn.Client.callback_cancel

```
import pysvn

cancel_command = False
def cancel():
    return cancel_command

client = pysvn.Client()
client.callback_cancel = cancel
```

The callback_cancel function is called frequently during long running commands. Return True to cause the command to cancel, return False to allow the command to continue.

pysvn.Client.callback_get_log_message

```
import pysvn

log_message = "reason for change"
def get_log_message():
    return rc, log_message

client = pysvn.Client()
client.callback_get_log_message = get_log_message
```

The callback_get_log_message is called when a log message is required to complete the current command. Return the True in rc and a log message as a string. Returning False in rc will cause the command to be cancelled. An empty log_message is not allowed and may cause the command to be cancelled.

Unicode strings cannot be handled. If you have a unicode string, convert it to UTF-8.

pysvn.Client.callback_get_login

```
import pysvn

def get_login( realm, username, may_save ):
    return retcode, username, password, save

client = pysvn.Client()
client.callback_get_login = get_login
```

callback_get_login is called each time subversion needs a username and password in the realm to access a repository and has no cache credentials.

The may_save parameter is true if subversion is willing to save the answers returned by the callback_get_login function.

pysvn expect the callback_get_login to return a tuple of four values (retcode, username, password, save).

- retcode - boolean, False if no username and password are available. True if subversion is to use the username and password.
- username - string, the username to use
- password - string, the password to use
- save - boolean, return True if you want subversion to remember the username and password in the configuration directory. return False to prevent saving the username and password.

pysvn.Client.callback_notify

```
import pysvn

def notify( event_dict ):
    return

client = pysvn.Client()
client.callback_notify = notify
```

The callback_notify is called as a command runs each time an interesting event occurs. The details of the event are passed to the callback_notify function as a dictionary.

The dictionary contains the following values:

- path - the path of the action refers to
- action - the events action, one of the wc_notify_action values
- kind - the node kind, one of the pysvn.node_kind values
- mime_type - the mime type
- content_state - one of the pysvn.wc_notify_state values
- prop_state - one of the pysvn.wc_notify_state values
- revision - a Revision object

pysvn.Client.callback_ssl_client_cert_password_prompt

```
import pysvn

def ssl_client_cert_password_prompt( realm, may_save ):
    return retcode, password, save

client = pysvn.Client()
client.callback_ssl_client_cert_password_prompt = ssl_client_cert_password_prompt
```

callback_ssl_client_cert_password_prompt is called each time subversion needs a password in the realm to use a client certificate and h no cached credentials.

The may_save parameter is true if subversion is willing to save the answers returned by the callback_ssl_client_cert_password_prompt function.

pysvn expect the callback_ssl_client_cert_password_prompt to return a tuple of three values (retcode, password, save).

- retcode - boolean, False if no password is available. True if subversion is to use password.
- password - string, the password to use
- save - boolean, return True if you want subversion to remember the password in the configuration directory. return False to preven saving the password.

pysvn.Client.callback_ssl_client_cert_prompt

```
import pysvn

def ssl_client_cert_prompt( realm, may_save ):
    return retcode, certfile, may_save

client = pysvn.Client()
client.callback_ssl_client_cert_prompt = ssl_client_cert_prompt
```

callback_ssl_client_cert_prompt is called each time subversion needs a client certificate.

pysvn expect the callback_ssl_client_cert_prompt to return a tuple of three values (retcode, certfile, may_save).

- retcode - boolean, False if no certificate is available. True if subversion is to use the certificate.
- certfile - string, the certfile to use

pysvn.Client.callback_ssl_server_prompt

```
import pysvn

def ssl_server_prompt( ):
    return

client = pysvn.Client()
client.callback_ssl_server_prompt = ssl_server_prompt
```

NOT IMPLEMENTED - what it used for?

pysvn.Client.callback_ssl_server_trust_prompt

```
import pysvn

def ssl_server_trust_prompt( trust_dict ):
    return retcode, accepted_failures, save

client = pysvn.Client()
client.callback_ssl_server_trust_prompt = ssl_server_trust_prompt
```

The callback_ssl_server_trust_prompt is called each time an HTTPS server presents a certificate and subversion is not sure if it should l trusted. callback_ssl_server_trust_prompt is called with information about the certificate in trust dict.

- failures - int - a bitmask of failures - [What do these bits mean?]
- hostname - string - the hostname the certificate was presented from
- finger_print - string - certificate finger print
- valid_from - string - valid from this ISO8601 date
- valid_until - string - valid util this ISO8601 date
- issuer_dname - stirng - the issued dname
- realm - string - the realm

pysvn expect the callback_ssl_server_trust_prompt to return a tuple of three values (retcode, accepted_failures, save).

- retcode - boolean, True if the ssl server is trusted, False if not trusted.
- accepted_failures - int, the accepted failures allowed. Typically just return trust_dict["failures"].
- save - boolean, return True if you want subversion to remember the certificate in the configuration directory. return False to prever saving the certificate.

pysvn.Client.callback_conflict_resolver

```
import pysvn

def conflict_resolver( conflict_description ):
    return conflict_choice, merge_file, save_merged

client = pysvn.Client()
client.callback_conflict_resolver = conflict_resolver
```

The callback_conflict_resolver is called each time a conflict needs resolving. It is passed the conflict_description and must return a conflict_choice, merge_file and save_merged.

The members of the conflict_description dictionary are:

- path - string - The path that is in conflict (for a tree conflict, it is the victim)
- node_kind - pysvn.node_kind - The node type of the path being operated on
- kind - pysvn.conflict_kind - the sort of conflict being described
- property_name - string or None - The name of the property whose conflict is being described. (Only if kind is 'property'; else undefined.)
- is_binary - boolean - Whether svn thinks ('my' version of) path is a 'binary' file. (Only if kind is 'text', else undefined.)
- mime_type - string or None - The svn:mime-type property of ('my' version of) path
- action - pysvn.wc_conflict_action - the action being attempted on the conflicted node or property
- reason - pysvn.wc_conflict_reason - The state of the target node or property, relative to its merge-left source, that is the reason fo the conflict
- base_file - string - common ancestor of the two files being merged
- their_file - string - their version of the file
- my_file - string - my locally-edited version of the file
- merged_file - string - merged version; may contain conflict markers
- operation - pysvn.wc_operation - the operation that exposed the conflict. Used only for tree conflicts
- src_left_version - pysvn.wc_conflict_version - Info on the "merge-left source" or "older" version of incoming change
- src_right_version - pysvn.wc_conflict_version - Info on the "merge-right source" or "their" version of incoming change

- conflict_choice is one of the pysvn.wc_conflict_choice values
- merge_file is a file name or None

- save_merged is True or False

Client methods

add	add_to_changelist	annotate	annotate2	cat	checkin
checkout	cleanup	copy	copy2	diff	diff_peg
diff_summarize	diff_summarize_peg	export	get_adm_dir	get_auth_cache	get_auto_props
get_changelist	get_default_password	get_default_username	get_interactive	get_store_passwords	import_
info	info2	is_adm_dir	is_url	list	log
lock	ls	merge	merge_peg	merge_peg2	merge_peg2
mkdir	move	move2	patch	propdel	propdel_local
propdel_remote	propget	proplist	propset	propset_local	propset_remote
relocate	remove	remove_from_changelists	resolved	revert	revpropdel
revpropget	revproplist	revpropset	root_url_from_path	set_adm_dir	set_auth_cache
set_auto_props	set_default_password	set_default_username	set_interactive	set_store_passwords	status2
status	switch	unlock	update	upgrade	vacuum

pysvn.Client.add_to_changelist

```
add_to_changelist( path,
                   changelist,
                   depth=QQQ,
                   changelists=[] )
```

TBD

depth is one of the pysvn.depth enums.

pysvn.Client.add

```
add( path,
     recurse=True,
     force=False,
     ignore=False,
     depth=None,
     add_parents=False,
     autoprops=False )
add( [path,path],
     recurse=True,
     force=False,
     ignore=True,
     depth=None,
     add_parents=False,
     autoprops=True )
```

Schedules all the files or directories specfied in the paths for addition to the repository. Set recurse to True to recursively add a directory's children. Set force to True to force operation to run. Set ignore to False to disregard default and svn:ignore property ignores Files are added at the next checkin.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

Set add_parents True to have subversion create missing directories.

Set autoprops to False to prevent subversion from applying properties automatically.

pysvn.Client.annotate

```
file_annotation = \
annotate( url_or_path,
          revision_start=pysvn.Revision( opt_revision_kind.number, 0 ),
          revision_end=pysvn.Revision( opt_revision_kind.head ),
          peg_revision=pysvn.Revision( opt_revision_kind.unspecified ) )
 )
```

Return the annotation for each line in the url_or_path from revision_start to revision_end.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The file_annotation is a list of dictionaries. Each dictionary contains:

- author - string - the name of the author who last wrote the line
- date - string - the date of the last change to the line
- line - string - the text of the line
- number - int - the line number

- revision - pysvn.Revision - the revision that committed the line

pysvn.Client.annotate2

```
file_annotation = \
annotate( url_or_path,
          revision_start=pysvn.Revision( opt_revision_kind.number, 0 ),
          revision_end=pysvn.Revision( opt_revision_kind.head ),
          peg_revision=pysvn.Revision( opt_revision_kind.unspecified ) )
 )
```

Return the annotation for each line in the url_or_path from revision_start to revision_end.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The file_annotation is a list of dictionaries. Each dictionary contains:

- line - string - the text of the line
- number - int - the line number
- revision - pysvn.Revision - the revision that committed the line
- local_change - boolean - true if the change was made locally
- merged_revision - pysvn.Revsion or None - None if not merged otherwise revision of the merged line
- merged_path - string or None - path of the merged line if merged
- local_changed - boolean - If there is no blame information for this line, revision will be invalid and rev_props will be None. In this case local_change will be True if the reason there is no annotation information is that the line was modified locally. In all other case local_change will be False.
- merged_rev_props - not implemented yet

Note: To find the author and date of the lines in the annotation use log().

pysvn.Client.cat

```
file_text = \
cat( url_or_path,
     revision=pysvn.Revision( opt_revision_kind.head ),
     peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
     expand_keywords=false,
     get_props=false )


file_text, props = \
cat( url_or_path,
     revision=pysvn.Revision( opt_revision_kind.head ),
     peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
     expand_keywords=False,
     get_props=True )
```

Return the contents of url_or_path for the specified revision as a string, file_text.

When get_props is True the props of the file are returned in a dictionary.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

pysvn.Client.checkin

```
revision = \
checkin( path,
        log_message,
        recurse=True,
        keep_locks=False,
        depth,
        keep_changelist,
        changelists,
        revprops,
        commit_as_operations=False,
        include_file_externals=False,
        include_dir_externals=False )
revision = \
checkin( [path,path],
        log_message,
        recurse=True,
        keep_locks=False,
```

```
        depth,
        keep_changelist,
        changelists,
        revprops,
        commit_as_operations=False,
        include_file_externals=False,
        include_dir_externals=False )
```

checkin the files in the path_list to the repository with the specified log_message. Set recurse to True to recursively checkin a directory children with the same log message. Set keep_locks to True to prevent locks in the path being unlocked.

checkin returns a pysvn.Revision containing the number of the checked in revision.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

changelists is an array of string changelist names, used as a restrictive filter on items that are committed; that is, don't commit anythir unless it's a member of one of those changelists. After the commit completes successfully, remove changelist associations from the targets, unless keep_changelists is set. If changelists is empty or None, no changelist filtering occurs.

If not None revprop is a list holding additional, custom revision properties to be set on the new revision. This list cannot contain any standard Subversion properties.

If commit_as_operations is set to False, when a copy is committed all changes below the copy are always committed at the same time (independent of the value of depth). If commit_as_operations is True, changes to descendants are only committed if they are itself included via depth and targets.

If include_file_externals and/or include_dir_externals are True, also commit all file and/or dir externals (respectively) that are reached recursion, except for those externals which:

- have a fixed revision, or
- come from a different repository root URL (dir externals).

These flags affect only recursion; externals that directly appear in targets are always included in the commit.

pysvn.Client.checkout

```
revision = \
checkout( url,
          path,
          recurse=True,
          revision=pysvn.Revision( opt_revision_kind.head ),
          peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
          ignore_externals=False,
          allow_unver_obstructions=False,
          depth=None )
```

checkout the repository at url into the location specified by path. Set recurse to True to recursively check out a directory's children. Specify a revision to check out a particular version of the source tree. Set ignore_externals to True to ignore externals definitions.

peg_revision indicates in which revision url is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

checkout returns a pysvn.Revision containing the number of the checked out revision.

Note: Subversion seems to return 0 rather then the actual revision. Use a notify callback and record the revision reported for the pysvn.wc_notify_action.update_completed event. This is what the svn command does.

pysvn.Client.cleanup

```
cleanup( path,
         fix_recorded_timestamps=True,
         clear_dav_cache=True,
         vacuum_pristines=True,
         include_externals=False )
```

Clean up any locks in the working copy at path. Usually such locks are the result of a failed or interrupted operation.

If break_locks is True, existing working copy locks at or below dir_abspath are broken, otherwise a normal write lock is obtained.

If fix_recorded_timestamps is True, this function fixes recorded timestamps for unmodified files in the working copy, reducing comparision time on future checks.

If clear_dav_cache is True, the caching of DAV information for older mod_dav served repositories is cleared. This clearing invalidates some cached information used for pre-HTTPv2 repositories.

If vacuum_pristines is True, and dir_abspath points to the working copy root unreferenced files in the pristine store are removed.

If include_externals is True, recurse into externals and clean them up as well.

pysvn.Client.copy

```
copy( src_url_or_path,
      dest_url_or_path,
      src_revision=pysvn.Revision( opt_revision_kind.head )
      )
```

Duplicate something in working copy or repos, remembering history. The src_revision defaults to pysvn.Revision(opt_revision_kind.hea) if the src_path is a URL otherwise to pysvn.Revision(opt_revision_kind.working).

src_url_or_path and dest_url_or_path can each be either a working copy (WC) path or URL:

- WC -&rt; WC: copy and schedule for addition (with history)
- WC -&rt; URL: immediately commit a copy of WC to URL
- URL -&rt; WC: check out URL into WC, schedule for addition
- URL -&rt; URL: complete server-side copy; used to branch and tag

If the destination is a URL the client_get_log_message callback must be implemented to return a log message.

pysvn.Client.copy2

```
copy2( sources,
       dest_url_or_path,
       copy_as_child=False,
       make_parents=False,
       revprops,
       ignore_externals=False,
       metadata_only=False,
       pin_externals=False,
       externals_to_pin=[(path_or_url, externals_description),...] )
```

Duplicate something in working copy or repos, remembering history. The src_revision defaults to pysvn.Revision(opt_revision_kind.hea) if the src_path is a URL otherwise to pysvn.Revision(opt_revision_kind.working).

sources is a list of tuples of (url_or_path, rev), you can ommit rev by passing (url_or_path,). TBD better docs here.

revprops TBD

set ignore_externals to True to ignore externals.

If metadata_only is True and copying a file in a working copy, everything in the metadata is updated as if the node is moved, but the actual disk copy operation is not performed. This feature is useful for clients that want to keep the working copy in sync while the actua working copy is updated by some other task.

If pin_externals is set, pin URLs in copied externals definitions to their current revision unless they were already pinned to a particular revision. A pinned external uses a URL which points at a fixed revision, rather than the HEAD revision. Externals in the copy destination are pinned to either a working copy base revision or the HEAD revision of a repository (as of the time the copy operation is performed), depending on the type of the copy source:

If not None, externals_to_pin restricts pinning to a subset of externals. It is a dictionary keyed by either a local absolute path or a URL which an svn:externals property is set. The dictionary contains externals description each of which corresponds to a single line of an svn:externals definition.

Externals corresponding to these items will be pinned, other externals will not be pinned. If externals_to_pin is None then all externals are pinned. If pin_externals is False then externals_to_pin is ignored.

src_url_or_path and dest_url_or_path can each be either a working copy (WC) path or URL:

- WC -> WC: copy and schedule for addition (with history)
- WC -> URL: immediately commit a copy of WC to URL
- URL -> WC: check out URL into WC, schedule for addition
- URL -> URL: complete server-side copy; used to branch and tag

If the destination is a URL the client_get_log_message callback must be implemented to return a log message.

pysvn.Client.diff

```
diff_text = \
diff( tmp_path,
      url_or_path,
      revision1=pysvn.Revision( opt_revision_kind.base ),
      url_or_path2=url_or_path,
      revision2=pysvn.Revision( opt_revision_kind.working ),
      recurse=True,
      ignore_ancestry=False,
      diff_deleted=True,
      ignore_content_type=False,
      header_encoding="",
      diff_options=[],
```

```
      depth=depth,,
      relative_to_dir=None,
      changelists=None,
      show_copies_as_adds=False,
      use_git_diff_format=False,
      diff_added=False,
      ignore_properties=False,
      properties_only=False )
```

Return the differences between revision1 of url_or_path and revision2 of url_or_path2. diff_text is a string containing the diff output.

diff uses tmp_path to form the filename when creating any temporary files needed. The names are formed using tmp_path + unique_string + ".tmp". For example tmp_path=/tmp/diff_prefix will create files like /tmp/diff_prefix.tmp and /tmp/diff_prefix1.tmp.

Diff output will not be generated for binary files, unless ignore_content_type is true, in which case diffs will be shown regardless of the content types.

Generated headers are encoded using header_encoding.

The list of diff_options strings are passed to the external diff program that subversion uses. Typical options are -b (ignore space change and -w (ignore all white space). The exact options that work depend on the version of subversion used and its configuration.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

If relative_to_dir is not None, the original_path and modified_path will have the relative_to_dir stripped from the front of the respective paths.

If relative_to_dir is not None but relative_to_dir is not a parent path of the target, an error is returned.

If show_copies_as_adds is True, then copied files will not be diffed against their copyfrom source, and will appear in the diff output in their entirety, as if they were newly added.

If use_git_diff_format is True, then the git's extended diff format will be used.

if diff_added is True show diff of added files.

if ignore_properties is True then ignore diff of properties.

if properties_only is True then only diff properties.

pysvn.Client.diff_peg

```
diff_text = \
diff_peg( tmp_path,
          url_or_path,
          peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
          revision_start=pysvn.Revision( opt_revision_kind.base ),
          revision_end=pysvn.Revision( opt_revision_kind.working ),
          recurse=True,
          ignore_ancestry=False,
          diff_deleted=True,
          ignore_content_type=False,
          header_encoding="",
          diff_options=[],
          depth=depth,
          show_copies_as_adds=False,
          use_git_diff_format=False,
          diff_added=False,
          ignore_properties=False,
          properties_only=False )
```

return the differences between two revisions of the url_or_path. diff_text is a string containing the diff output.

diff uses tmp_path to form the filename when creating any temporary files needed. The names are formed using tmp_path + unique_string + ".tmp". For example tmp_path=/tmp/diff_prefix will create files like /tmp/diff_prefix.tmp and /tmp/diff_prefix1.tmp.

Set recurse to True to recursively diff a directory's children. diff_text is a string containing the diff.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

Diff output will not be generated for binary files, unless ignore_content_type is true, in which case diffs will be shown regardless of the content types.

Generated headers are encoded using header_encoding.

The list of diff_options strings are passed to the external diff program that subversion uses. Typical options are -b (ignore space change and -w (ignore all white space). The exact options that work depend on the version of subversion used and its configuration.

If show_copies_as_adds is True, then copied files will not be diffed against their copyfrom source, and will appear in the diff output in their entirety, as if they were newly added.

If use_git_diff_format is True, then the git's extended diff format will be used.

if diff_added is True show diff of added files.

if ignore_properties is True then ignore diff of properties.

if properties_only is True then only diff properties.

pysvn.Client.diff_summarize

```
summary = \
diff_summarize( url_or_path1,
                revision1=pysvn.Revision( opt_revision_kind.base ),
                url_or_path2=url_or_path,
                revision2=pysvn.Revision( opt_revision_kind.working ),
                recurse=True,
                ignore_ancestry=False,
                depth=depth )
```

Produce a diff summary which lists the changed items between url_or_path1 revision1 and url_or_path2 revision2 without creating text deltas. url_or_path1 and url_or_path2 can be either working-copy paths or URLs.

The function may report false positives if ignore_ancestry is False, since a file might have been modified between two revisions, but stil have the same contents.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

pysvn.Client.diff_summarize_peg

```
diff_text = \
diff_summarize_peg( url_or_path,
                    peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
                    revision_start=pysvn.Revision( opt_revision_kind.base ),
                    revision_end=pysvn.Revision( opt_revision_kind.working ),
                    recurse=True,
                    ignore_ancestry=False,
                    depth=depth )
```

Produce a diff summary which lists the changed items between the filesystem object url_or_path in peg revision peg_revision, as it changed between revision_start and revision_end. url_or_path can be either a working-copy path or URL.

If peg_revision is opt_revision_unspecified, behave identically to svn_client_diff_summarize(), using path for both of that function's url_or_path1 and url_or_path2 arguments.

The function may report false positives if ignore_ancestry is False, as described in the documentation for diff_summarize().

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

pysvn.Client.export

```
revision = \
export( src_url_or_path,
        dest_path,
        force=False,
        revision=pysvn.Revision(),
        native_eol=None,
        ignore_externals=False,
        recurse=True,
        peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
        depth=depth,
        ignore_keywords=False )
```

Create an unversioned copy of the src_path at revision in dest_path. Set recurse to False to export a single file. Set ignore_externals tc True to ignore externals definitions. Set ignore_keywords to True to prevnet keyword replacement.

peg_revision indicates in which revision src_url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults t opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

1. Exports a clean directory tree from the repository specified by URL src_url_or_path, at revision if it is given, otherwise at HEAD, into dest_path.
2. Exports a clean directory tree from the working copy specified by src_path into dest_path. All local changes will be preserved, bu files not under revision control will not be copied.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

native_eol parameter allows the line ending of files with svn:eol-style property set to native to be overridden. Use None to use the eol-style of the Operating System, use "LF" to use "\n", "CR" to use "\r" and "CRLF" to use "\r\n".

export returns a pysvn.Revision containing the number of the checked in revision.

Note: The native_eol parameter is only available for svn 1.1.0 or later.

pysvn.Client.get_auth_cache

```
enabled = get_auth_cache()
```

return true if credential caching is enabled, otherwise return false.

pysvn.Client.get_adm_dir

```
name = get_adm_dir()
```

Returns the name of the subverion admin directory.

pysvn.Client.get_auto_props

```
enabled = get_auto_props()
```

Returns true if svn will automatically set properties when adding files, otherwise returns false.

pysvn.Client.get_changelist

```
changelist_list = \
get_changelist( path,
                changelist,
                depth=depth,
                changelists=[] )
```

TBD

The depth is one of the pysvn.depth enums.

pysvn.Client.get_default_password

```
password = get_default_password()
```

Returns None if no default is set otherwise returns the password as a string.

pysvn.Client.get_default_username

```
username = get_default_username()
```

Returns None if no default is set otherwise returns the username as a string.

pysvn.Client.get_interactive

```
enabled = get_interactive()
```

Returns true if svn will prompt for missing credential information, otherwise returns false.

pysvn.Client.get_store_passwords

```
enabled = get_store_passwords()
```

Returns true if svn will store passwords after prompting for them, otherwise returns false.

pysvn.Client.import_

```
revision = \
import_( path,
         url,
         log_message,
         recurse=True,
         ignore=False,
         revprops,
         autoprops=False )
```

Commit an unversioned file or tree into the repository.

Recursively commit a copy of PATH to URL. Parent directories are created as necessary in the repository. Set ignore to False to disregar
default and svn:ignore property ignores.

revprops TBD

import_ returns a pysvn.Revision containing the number of the checked in revision.

pysvn.Client.info

```
entry = info( path )
```

return information on path as a Entry object.

Set autoprops to False to prevent subversion from applying properties automatically.

pysvn.Client.info2

```
entry_list = \
info2( url_or_path,
       revision=pysvn.Revision( opt_revision_kind.unspecified ),
       peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
       recurse=True,
       depth=depth
       fetch_excluded=False,
       fetch_actual_only=True )
```

return information on url_or_path as a list of (path, info_dict) tuples. To return information about a URL revision must be opt_revision_kind.head or opt_revision_kind.number.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

If fetch_excluded is True, also also send excluded nodes in the working copy to receiver, otherwise these are skipped.

If fetch_actual_only is True also send nodes that don't exist as versioned but are still tree conflicted.

The info_dict contains:

- URL - URL or path
- rev - pysvn.Revision or None
- kind - kind of path
- repos_root_URL - string
- repos_UUID - string
- last_changed_rev - pysvn.Revision or None
- last_changed_date - time or None
- last_changed_author - string or None
- lock - None or dictionary containing:
  - path - string
  - token - string or None
  - owner - string or None
  - comment - string or None
  - is_dav_comment - true if is DAV comment
  - creation_date - time or None
- wc_info - None if url_or_path is a URL; otherwise a dictionary containing:
  - schedule - pysvn.wc_schedule or None
  - copyfrom_url - string or None
  - copyfrom_rev - pysvn.Revision or None
  - text_time - time or None
  - prop_time - time or None
  - checksum - string or None
  - conflict_old - string or None
  - conflict_new - string or None
  - conflict_wrk - string or None
  - prejfile - string or None

Note: The info2 command is only available with svn 1.2.0 or later.

pysvn.Client.is_adm_dir

```
rc = \
is_adm_dir( name )
```

Return True is the name is an subversion admin directory.

pysvn.Client.root_url_from_path

```
root_url = \
    root_url_from_path( url_or_path )
```

Return the root URL of the repository given the url_or_path.

pysvn.Client.is_url

```
rc = \
is_url( url )
```

return True if the url is a known subversion url.

pysvn.Client.list

```
entries_list = \
list( url_or_path,
      peg_revision=pysvn.Revision( opt_revision_kind.unspecified ) )
      revision=pysvn.Revision( opt_revision_kind.head ),
      recurse=True,
      dirent_fields=pysvn.SVN_DIRENT_ALL,
      fetch_locks=False,
      depth=depth )
```

Returns a list with a tuple of information for each file in the given path at the provided revision.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

dirent_fields controls which dirent fields will return. Use pysvn.SVN_DIRENT_ALL to return all fields. Bit-wise or one of these values to return only the selected fields:

- SVN_DIRENT_KIND - return kind field
- SVN_DIRENT_SIZE - return size field
- SVN_DIRENT_HAS_PROPS - return has_props field
- SVN_DIRENT_CREATED_REV - return created_rev field
- SVN_DIRENT_TIME - return time field
- SVN_DIRENT_LAST_AUTHOR - return last_author field

The tuple contains:

- 0 - PysvnList containing the dirent information
- 1 - PysvnLock containing the lock information or None
- 2 - external_parent_url - string or None
- 3 - external_target - string or None

The PysvnList object contains the requested dirent fields:

- created_rev - pysvn.Revision - the revision of the last change
- has_props - bool - True if the node has properties
- kind - node_kind - one of the pysvn.node_kind values
- last_author - string - the author of the last change
- repos_path - string - (always present) absolute path of file in the repository
- size - long - size of file
- time - float - the time of the last change

The PysvnList object obtains the lock information:

- comment - string - the lock comment
- token - string - lock token
- path -
- owner - string - owner of the lock
- expiration_date - None or int - time lock will expire
- is_dav_comment - boolean - true is a commment from DAV
- creation_date - int - time the lock was created

If list() was called with include_externals set to True, external_parent_url and external_target will be set. external_parent_url is url of t directory which has the externals definitions. external_target is the target subdirectory of externals definitions which is relative to the parent directory that holds the external item.

If external_parent_url and external_target are defined, the item being listed is part of the external described by external_parent_url an external_target. Else, the item is not part of any external. Moreover, we will never mix items which are part of separate externals, and will always finish listing an external before listing the next one.

pysvn.Client.lock

```
lock( url_or_path,
      lock_comment,
      force=False )
```

lock the url_or_path with lock_comment. Set force to True to override any lock held by another user.

pysvn.Client.log

```
log_messages = \
log( url_or_path,
     revision_start=pysvn.Revision( opt_revision_kind.head ),
     revision_end=pysvn.Revision( opt_revision_kind.number, 0 ),
     discover_changed_paths=False,
     strict_node_history=True,
     limit=0,
     peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
     include_merged_revisions=False,
     revprops=list_of_revprop_names )
```

Return the log messages for the specified url_or_path between revisions start and end. Set limit to the maximum number of log messages to be returned, 0 means return all messages.

If discover_changed_paths is set, the changed_paths dictionary entry is filled with a list of changed paths. If strict_node_history is set, log entries will not cross copies.

If url_or_path no longer exists in the repos of WC then pass in a peg_revision of a revision where it did exist.

include_merged_revisions TBD

revprops is a list of strings that name the revprops to be returned.

log returns a list of log entries; each log entry is a dictionary. The dictionary contains:

- author - string - the name of the author who committed the revision
- date - float time - the date of the commit
- message - string - the text of the log message for the commit
- revision - pysvn.Revision - the revision of the commit
- changed_paths - list of dictionaries. Each dictionary contains:
  - path - string - the path in the repository
  - action - string
  - copyfrom_path - string - if copied, the original path, else None
  - copyfrom_revision - pysvn.Revision - if copied, the revision of the original, else None

pysvn.Client.ls

```
entries_list = \
ls( url_or_path,
    revision=pysvn.Revision( opt_revision_kind.head ),
    recurse=True,
    peg_revision=pysvn.Revision( opt_revision_kind.unspecified ) )
```

Use the list method in new code as it fixes performance and ambiguity problems with the ls method.

Returns a list of dictionaries for each file the given path at the provided revision.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The dictionary contains:

- created_rev - pysvn.Revision - the revision of the last change
- has_props - bool - True if the node has properties
- kind - node_kind - one of the pysvn.node_kind values
- last_author - the author of the last change
- name - string - name of the file
- size - long - size of file
- time - float - the time of the last change

pysvn.Client.merge

```
merge( url_or_path1,
       revision1,
       url_or_path2,
       revision2,
       local_path,
```

```
        force=False,
        recurse=True,
        notice_ancestry=False,
        dry_run=False,,
        depth=depth,
        record_only=False,
        merge_options=[],
        allow_mixed_revisions=false,
        ignore_mergeinfo=not notice_ancestry )
```

Apply the differences between two sources to a working copy path.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

merge_options (a list of strings), is used to pass additional command line arguments to the merge processes (internal or external).

The internal subversion diff supports the following options:

- --ignore-space-change, -b
- --ignore-all-space, -w
- --ignore-eol-style
- --unified, -u (for compatibility, does nothing).

If allow_mixed_revisions is False, and merge_target is a mixed-revision working copy, raise SVN_ERR_CLIENT_MERGE_UPDATE_REQUIRED.

Because users rarely intend to merge into mixed-revision working copies, it is recommended to set this parameter to FALSE by default unless the user has explicitly requested a merge into a mixed-revision working copy.

If ignore_mergeinfo is true, disable merge tracking, by treating the two sources as unrelated even if they actually have a common ancestor.

pysvn.Client.merge_peg

```
merge_peg( url_or_path,
           revision1,
           revision2,
           peg_revision,
           local_path,
           recurse=True,
           notice_ancestry,
           force=False,
           dry_run=False,
           depth=depth,
           record_only=False,
           merge_options=[] )
```

Apply the differences between two sources to a working copy path.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

merge_options (a list of strings), is used to pass additional command line arguments to the merge processes (internal or external).

The internal subversion diff supports the following options:

- --ignore-space-change, -b
- --ignore-all-space, -w
- --ignore-eol-style
- --unified, -u (for compatibility, does nothing).

pysvn.Client.merge_peg2

```
merge_peg2( sources,
            ranges_to_merge,
            peg_revision,
            tareget_wcpath,
            depth=depth,
            notice_ancestry=False,
            force=False,
            dry_run=False,
            record_only=True,
            merge_options=[],
            allow_mixed_revisions=false,
            ignore_mergeinfo=false )
```

Apply the differences between the ranges_to_merge in sources to a working copy path, target_wcpath. ranges_to_merge is a list of tuples with the start and end revisions to be merged.

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The depth is one of the pysvn.depth enums.

merge_options (a list of strings), is used to pass additional command line arguments to the merge processes (internal or external).

The internal subversion diff supports the following options:

- --ignore-space-change, -b
- --ignore-all-space, -w
- --ignore-eol-style
- --unified, -u (for compatibility, does nothing).

If allow_mixed_revisions is False, and merge_target is a mixed-revision working copy, raise SVN_ERR_CLIENT_MERGE_UPDATE_REQUIRED.

Because users rarely intend to merge into mixed-revision working copies, it is recommended to set this parameter to FALSE by default unless the user has explicitly requested a merge into a mixed-revision working copy.

If ignore_mergeinfo is true, disable merge tracking, by treating the two sources as unrelated even if they actually have a common ancestor.

pysvn.Client.merge_reintegrate

```
merge_reintegrate( url_or_path,
                   revision,
                   local_path,
                   dry_run=False,
                   merge_options=[] )
```

Lump-merge all of url_or_path unmerged changes into local_path.

merge_options (a list of strings), is used to pass additional command line arguments to the merge processes (internal or external).

The internal subversion diff supports the following options:

- --ignore-space-change, -b
- --ignore-all-space, -w
- --ignore-eol-style
- --unified, -u (for compatibility, does nothing).

pysvn.Client.mkdir

```
mkdir( url_or_path,
       log_message,
       make_parents,
       revprops )
mkdir( [url_or_path,url_or_path],
       log_message,
       make_parents,
       revprops )
```

Create a new directory under revision control.

url_or_path can be a list of URLs and paths

make_parents and revprops TBD

If url_or_path is a path, each directory is scheduled for addition upon the next commit.

If url_or_path is a URL, the directories are created in the repository via an immediate commit.

In both cases, all the intermediate directories must already exist.

pysvn.Client.move

```
move( src_url_or_path,
      dest_url_or_path,
      force=False )
```

Move (rename) something in working copy or HEAD revision of repository.

NOTE: this command is equivalent to a 'copy' and 'delete'.

src_path and dest_path can both be working copy (WC) paths or URLs:

- WC -> WC: move and schedule for addition (with history)
- URL -> URL: complete server-side rename.

If src_url_or_path is a path, each item is scheduled for deletion upon the next commit. Files, and directories that have not been committed, are immediately removed from the working copy. The command will not remove PATHs that are, or contain, unversioned or modified items; set force=True to override this behaviour.

If src_url_or_path is a URL, the items are deleted from the repository via an immediate commit.

pysvn.Client.move2

```
move2( sources,
       dest_url_or_path,
       move_as_child=False,
       make_parents=False,
       revprops,
       allow_mixed_revisions=False,
       metadata_only=False )
```

Duplicate something in working copy or repos, remembering history. The src_revision defaults to pysvn.Revision(opt_revision_kind.hea) if the src_path is a URL otherwise to pysvn.Revision(opt_revision_kind.working).

If not None, revprop is a list of strings holding additional, custom revision properties to be set on the new revision in the event that this a committing operation. This table cannot contain any standard Subversion properties.

If allow_mixed_revisions is False, SVN_ERR_WC_MIXED_REVISIONS will be raised if the move source is a mixed-revision subtree.

If allow_mixed_revisions is True, a mixed-revision move source is allowed but the move will degrade to a copy and a delete without loc move tracking. This parameter should be set to False except where backwards compatibility to older versions of subversion is required.

If metadata_only is TRUE and moving a file in a working copy, everything in the metadata is updated as if the node is moved, but the actual disk move operation is not performed. This feature is useful for clients that want to keep the working copy in sync while the actu working copy is updated by some other task.

src_url_or_path and dest_url_or_path can each be either a working copy (WC) path or URL:

- WC -> WC: move and schedule for addition (with history)
- WC -> URL: immediately commit a move of WC to URL
- URL -> WC: check out URL into WC, schedule for addition
- URL -> URL: complete server-side move; used to branch and tag

If the destination is a URL the client_get_log_message callback must be implemented to return a log message.

pysvn.Client.patch

```
svn_client_patch(patch_abspath,
                 wc_dir_abspath,
                 dry_run=False,
                 strip_count=0,
                 reverse=False,
                 ignore_whitespace=False,
                 remove_tempfiles=False)
```

Apply a unidiff patch that's located at absolute path patch_abspath to the working copy directory at wc_dir_abspath.

This function makes a best-effort attempt at applying the patch. It might skip patch targets which cannot be patched (e.g. targets that are outside of the working copy). It will also reject hunks which cannot be applied to a target in case the hunk's context does not matcl anywhere in the patch target.

If dry_run is True, the patching process is carried out, and full notification feedback is provided, but the working copy is not modified.

strip_count specifies how many leading path components should be stripped from paths obtained from the patch. It is an error if a negative strip count is passed.

If reverse is True, apply patches in reverse, deleting lines the patch would add and adding lines the patch would delete.

If ignore_whitespace is True, allow patches to be applied if they only differ from the target by whitespace.

If remove_tempfiles is True, lifetimes of temporary files created during patching will be managed internally. Otherwise, the caller shoul take ownership of these files, the names of which can be obtained by passing a patch_func callback.

If notify_func is not None, invoke notify_func as patching progresses.

If cancel_func is not None, invoke it at various places during the operation.

pysvn.Client.propdel

```
rev = \
propdel( prop_name,
         url_or_path,
         revision=pysvn.Revision(),
         recurse=False,
         skip_checks=False,
```

```
        depth=depth,
        base_revision_for_url=pysvn.Revision( opt_revision_kind.number, see-text ),
        revprops )
```

Delete the property prop_name from url_or_path.

If skip_checks is true, do no validity checking. But if skip_checks is false, and propname is not a valid property for target, return an err either SVN_ERR_ILLEGAL_TARGET (if the property is not appropriate for target), or SVN_ERR_BAD_MIME_TYPE (if propname is "svn:mime-type", but propval is not a valid mime-type).

The url_or_path may only be an URL if base_revision_for_url is not -1; in this case, the property will only be set if it has not changed since revision base_revision_for_url. base_revision_for_url must be -1 if url_or_path is not an URL.

The src_revision defaults to pysvn.Revision(opt_revision_kind.head) if the src_path is a URL otherwise to pysvn.Revision(opt_revision_kind.working).

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

revprops TBD

pysvn.Client.propdel_local

```
propdel_local( prop_name,
               path,
               depth=empty,
               changelist=None )
```

path is either a string path or a list of path strings.

Delelte propname fromn each path in the list. The paths must be all working copy paths.

If depth is svn_depth_empty, set the property on each member of targets only; if svn_depth_files, set it on targets and their file childr (if any); if svn_depth_immediates, on targets and all of their immediate children (both files and directories); if svn_depth_infinity, on targets and everything beneath them.

changelists is an list of string changelist names, used as a restrictive filter on items whose properties are set; that is, don't set properti on any item unless it's a member of one of those changelists. If changelists is empty (or altogether None), no changelist filtering occurs

The context cancel callback can be used to cannle propset_local.

pysvn.Client.propdel_remote

```
commit_info = \
propdel_remote( prop_name,
                prop_value,
                url,
                base_revision_for_url=pysvn.Revision( opt_revision_kind.number, 0 ), )
```

Delete propname from the url.

Immediately attempt to commit the property change in the repository, using the log message returned from the context log messages callback.

If the property has changed on url since revision base_revision_for_url (must be opt_revision_kind.number), no change will be made a an error will be returned.

commit_info returns the detail of the successful commit.

pysvn.Client.propget

```
prop_list = \
propget( prop_name,
         url_or_path,
         revision=pysvn.Revision(),
         recurse=False,
         peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
         depth=depth,
         get_inherited_props=False )
```

Returns a dictionary with keys of url_or_path and values of the prop_name.

If get_inherited_props is True returns a tuple of two items:

- 0 - dictionary with keys of url_or_path and values of the prop_name.
- 1 - dictionary with keys of url_or_path and values of the prop_name that where inherited.

The src_revision defaults to pysvn.Revision(opt_revision_kind.head) if the url_or_path is a URL otherwise to pysvn.Revision(opt_revision_kind.working).

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

pysvn.Client.proplist

```
prop_list = \
proplist( url_or_path,
          revision=pysvn.Revision(),
          recurse=False,
          peg_revision=pysvn.Revision( opt_revision_kind.unspecified ),
          depth=depth )
```

Returns a list of tuples (path, prop_dict). The prop_dict contains the prop_names and their values if set on the path.

The src_revision defaults to pysvn.Revision(opt_revision_kind.head) if the url_or_path is a URL otherwise to pysvn.Revision(opt_revision_kind.working).

peg_revision indicates in which revision url_or_path is valid. If peg_revision.kind is opt_revision_kind.unspecified, then it defaults to opt_revision_kind.head for URLs or opt_revision_kind.working for WC targets.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

pysvn.Client.propset

```
commit_info = \
propset( prop_name,
         prop_value,
         url_or_path,
         revision=pysvn.Revision(),
         recurse=False,
         skip_checks=False,
         depth=depth,
         base_revision_for_url=[0 for URL, -1 for path],
         allow_unver_obstructions=False,
         revprops )
```

Set the property prop_name to prop_value in url_or_path.

If skip_checks is true, do no validity checking. But if skip_checks is false, and propname is not a valid property for target, return an err either SVN_ERR_ILLEGAL_TARGET (if the property is not appropriate for target), or SVN_ERR_BAD_MIME_TYPE (if propname is "svn:mime-type", but propval is not a valid mime-type).

The revision defaults to pysvn.Revision(opt_revision_kind.head) if the url_or_path is a URL otherwise to pysvn.Revision(opt_revision_kind.working).

The url_or_path may only be an URL if base_revision_for_url is not -1; in this case, the property will only be set if it has not changed since revision base_revision_for_url. base_revision_for_url must be -1 if url_or_path is not an URL.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

If allow_unver_obstructions is False then the update will abort if there are any unversioned obstructing items.

revprops TBD

pysvn.Client.propset_local

```
propset_local( prop_name,
               prop_value,
               path,
               depth=empty,
               skip_checks=False,
               changelist=None )
```

path is either a string path or a list of path strings.

Set propname to propval on each path in the list. The paths must be all working copy paths.

If depth is svn_depth_empty, set the property on each member of targets only; if svn_depth_files, set it on targets and their file childr (if any); if svn_depth_immediates, on targets and all of their immediate children (both files and directories); if svn_depth_infinity, on targets and everything beneath them.

changelists is an list of string changelist names, used as a restrictive filter on items whose properties are set; that is, don't set properti on any item unless it's a member of one of those changelists. If changelists is empty (or altogether None), no changelist filtering occurs

If propname is an svn-controlled property (i.e. prefixed with "svn:"), then the caller is responsible for ensuring that the value uses LF line-endings.

If skip_checks is True, do no validity checking. But if skip_checks is False, and propname is not a valid property for targets, return an error, either SVN_ERR_ILLEGAL_TARGET (if the property is not appropriate for targets), or SVN_ERR_BAD_MIME_TYPE (if propname is "svn:mime-type", but propval is not a valid mime-type).

The context cancel callback can be used to cannle propset_local.

pysvn.Client.propset_remote

```
commit_info = \
propset_remote( prop_name,
                prop_value,
                url,
                skip_checks=False,
                base_revision_for_url=0,
                revprops=None )
```

Set propname to propval on url.

Immediately attempt to commit the property change in the repository, using the log message returned from the context log messages callback.

If the property has changed on url since revision base_revision_for_url (which must not be SVN_INVALID_REVNUM), no change will be made and an error will be returned.

If non None, revprops is a list of tuples, (string names, string value), holding additional, custom revision properties to be set on the new revision. This list cannot contain any standard Subversion properties.

commit_info returns the detail of the successful commit.

If propname is an svn-controlled property (i.e. prefixed with "svn:"), then the caller is responsible for ensuring that the value uses LF line-endings.

If skip_checks is True, do no validity checking. But if skip_checks is False, and propname is not a valid property for url, return an error, either SVN_ERR_ILLEGAL_TARGET (if the property is not appropriate for url), or SVN_ERR_BAD_MIME_TYPE (if propname is "svn:mime-type", but propval is not a valid mime-type).

pysvn.Client.relocate

```
relocate( from_url,
          to_url,
          path,
          recurse=True,
          ignore_externals=True )
```

Relocate the working copy from from_url to to_url of path.

Set ignore_externals to False to work on the externals.

Note: recurse is ignored since subversion 1.7

pysvn.Client.remove

```
remove( url_or_path_list,
        force=False,
        keep_local=False,
        revprops )
```

If url_or_path is a path, each item is scheduled for deletion upon the next commit. Files, and directories that have not been committed, are immediately removed from the working copy. The command will not remove paths that are, or contain, unversioned or modified items; set force=True to override this behaviour.

Set keep_local to True to prevent the local file from being delete.

revprops TBD

If url_or_path is a URL, the items are deleted from the repository via an immediate commit.

pysvn.Client.remove_from_changelists

```
remove_from_changelists( path,
                         changelist,
                         depth=pysvn.depth.infinite,
                         changelists=[] )
```

TBD

pysvn.Client.resolved

```
resolved( path,
          recurse=True,
          depth=depth )
```

Mark the conflicted file at path as resolved.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

pysvn.Client.revert

```
revert( path,
        recurse=False,
        depth=depth,
        changelists=[],
        clear_changelists=false,
        metadata_only=false )
revert( [path,path],
        recurse=False,
        depth=depth,
        changelists=[],
        clear_changelists=false,
        metadata_only=false )
```

Discard any changes in the working copy at path. Set recurse to True to recursively revert a directory's children.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

pysvn.Client.revpropdel

```
rev = \
revpropdel( prop_name,
            url,
            revision=pysvn.Revision( opt_revision_kind.head ),
            force=False,
            original_prop_value=None )
```

Delete the revision property prop_name from url at the revision.

If original_prop_value is specified its string value is compared to the current value of the property and only if it matches does the revpropdel succeed.

pysvn.Client.revpropget

```
rev_prop = \
revpropget( prop_name,
            url,
            revision=pysvn.Revision( opt_revision_kind.head ) )
```

Returns a tuple (rev, prop_val) where the prop_val contains the revision property value.

pysvn.Client.revproplist

```
rev_prop_dict = \
revproplist( url,
             revision=pysvn.Revision( opt_revision_kind.head ) )
```

Returns a tuple (revision, prop_dict) where the prop_dict contains the revision properies and their values.

pysvn.Client.revpropset

```

old_prop_value = \

revpropset( prop_name,
            prop_value,
            url,
            revision=pysvn.Revision( opt_revision_kind.head ),
            force=False,
            original_prop_value=None )
```

set the revision property prop_name to prop_value in path. The old prop value is returned.

If original_prop_value is specified its string value is compared to the current value of the property and only if it matches does the revpropset succeed.

pysvn.Client.set_adm_dir

```
set_adm_dir( name )
```

Set the name of the subverion admin directory. ".svn" is the normal admin dir use and "_svn" is used on Windows to work around problems with .NET.

pysvn.Client.set_auth_cache

```
set_auth_cache( enable )
```

When enable is True subversion will remember authentication credentials in the configuration directory.

pysvn.Client.set_auto_props

```
set_auto_props( enable )
```

When enabled, subversion will automatically set properties when adding files; otherwise when disabled it will not.

pysvn.Client.set_default_password

```
set_default_password( password )
```

Set the default password to be used if there is no stored password.

pysvn.Client.set_default_username

```
set_default_username( username )
```

Set the default username to be used if there is no stored username.

pysvn.Client.set_interactive

```
set_interactive( enable )
```

When enable is True subversion will prompt for authentication credentials when there are no valid store credentials.

pysvn.Client.set_store_passwords

```
set_store_passwords( enable )
```

When enable is True subversion will store any passwords that subversion prompted for.

pysvn.Client.status2

```
status_list = \
status2( path,
         recurse=True,
         get_all=True,
         update=False,
         ignore=False,
         ignore_externals=False,
         depth=depth,
         changelist=[],
         depth_as_sticky=False,
         check_out_of_date=update,
         check_working_copy=True )
```

If path is a directory status is returned for all files in the directory in status_list. If path is a single file status is returned for that single f in status_list. Set ignore_externals to True to ignore externals definitions.

The status_list is a list of PysvnStatus objects.

Options:

- recurse - If recurse is True, recurse fully, else do only immediate children.
- get_all - If get_all is True, retrieve all entries; otherwise, retrieve only "interesting" entries (local mods and/or out-of-date).
- update - If update is set, contact the repository and augment the status structures with information about out-of-dateness.
- ignore - If ignore is False, the item will be added regardless of whether it is ignored.

- ignore_externals - If ignore_externals is False, then recurse into externals definitions (if any exist) after handling the main target. This calls the client notification function with the wc_notify_action.external action before handling each externals definition, and wit wc_notify_action.completed after each.
- depth - can be used as in place of recurse. depth is one of the pysvn.depth enums.
- If check_out_of_date is set, contact the repository and augment the status structures with information about out-of-dateness (with respect to revision). Also, if result_rev is not NULL, set *result_rev to the actual revision against which the working copy was compared (result_rev is not meaningful unless check_out_of_date is True). check_out_of_date deafults to the value of update.
- If check_working_copy is not set, do not scan the working copy for local modifications. This parameter will be ignored unless check_out_of_date is set. When set, the status report will not contain any information about local changes in the working copy; this includes local deletions and replacements.

pysvn.Client.status

```
status_list = \
status( path,
        recurse=True,
        get_all=True,
        update=False,
        ignore=False,
        ignore_externals=False,
        depth=depth )
```

If path is a directory status is returned for all files in the directory in status_list. If path is a single file status is returned for that single f in status_list. Set ignore_externals to True to ignore externals definitions.

The status_list is a list of PysvnStatus objects.

Options:

- recurse - If recurse is True, recurse fully, else do only immediate children.
- get_all - If get_all is True, retrieve all entries; otherwise, retrieve only "interesting" entries (local mods and/or out-of-date).
- update - If update is set, contact the repository and augment the status structures with information about out-of-dateness.
- ignore - If ignore is False, the item will be added regardless of whether it is ignored.
- ignore_externals (svn 1.2.0 or later) - If ignore_externals is False, then recurse into externals definitions (if any exist) after handlir the main target. This calls the client notification function with the wc_notify_action.external action before handling each externals definition, and with wc_notify_action.completed after each.
- depth - can be used as in place of recurse. depth is one of the pysvn.depth enums.

pysvn.Client.switch

```
switch( path,
        url,
        recurse=True,
        revision=pysvn.Revision( opt_revision_kind.head ),
        depth=depth,
        peg_revision=revision,
        depth_is_sticky=False,
        ignore_externals=False,
        allow_unver_obstructions=False )
```

Update the working copy to a different URL.

The depth can be used as in place of recurse. depth is one of the pysvn.depth enums.

If depth_is_sticky is set and depth is not svn_depth_unknown, then in addition to switching PATH, also set its sticky ambient depth valu to depth.

If ignore_externals is set, do not process externals definitions as part of this operation.

If allow_unver_obstructions is True then the switch tolerates existing unversioned items that obstruct added paths. Only obstructions o the same type (file or dir) as the added item are tolerated. The text of obstructing files is left as-is, effectively treating it as a user modification after the switch. Working properties of obstructing items are set equal to the base properties.

If allow_unver_obstructions is False then the switch will abort if there are any unversioned obstructing items.

1. Update the working copy to mirror a new URL. This behaviour is a superset of "svn update". Note: this is the way to move a working copy to a new branch.
2. Reconnect the working copy when the repository URL has changed.

pysvn.Client.unlock

```
unlock( url_or_path,
        force=False )
```

Unlock the url_or_path. Set force to True to unlock any lock held by another user.

pysvn.Client.update

```
revision = \
update( path,
        recurse=True,
        revision=pysvn.Revision( opt_revision_kind.head ),
        ignore_externals=False,
        depth=depth,
        depth_is_sticky=False,
        allow_unver_obstructions,
        adds_as_modifications=False,
        make_parents=False )
```

Update the file in the working copy at path to the specified revision. Set recurse to True to recursively update a directory's children. Set ignore_externals to True to ignore externals definitions.

path can be a single path string or a list of path strings.

The depth can be used in place of recurse. depth is one of the pysvn.depth enums. Use pysvn.depth.unknown to update all files and folders in the working copy honoring the current depths. Use pysvn.depth.infinity to upadate all files and folders adding any that are missing ignoring the current depths.

If depth_is_sticky is set and depth is not svn_depth_unknown, then in addition to updating PATHS, also set their sticky ambient depth value to depth.

If allow_unver_obstructions is True then the update tolerates existing unversioned items that obstruct added paths. Only obstructions c the same type (file or dir) as the added item are tolerated. The text of obstructing files is left as-is, effectively treating it as a user modification after the update. Working properties of obstructing items are set equal to the base properties. If allow_unver_obstructions False then the update will abort if there are any unversioned obstructing items.

If adds_as_modification is True, a local addition at the same path as an incoming addition of the same node kind results in a normal no with a possible local modification, instead of a tree conflict.

If make_parents is True, create any non-existent parent directories also by checking them out at depth=empty.

update returns a pysvn.Revision containing the number of the revision the working copy was updated to.

This command is typically used to get the latest changes from the repository.

Note: updating to an older revision does not change the current revision. To make the current version identical to an older revision, use merge followed by a commit.

pysvn.Client.upgrade

```
upgrade( path )
```

Recursively upgrade a working copy from any older format to the current WC metadata storage format. path is the path to the WC root

pysvn.Transaction - Subversion transaction interface

Interface summary:

```
transaction = pysvn.Transaction()
transaction = pysvn.Transaction( repos_path, transaction_name, [is_revision=False] )
```

The Transaction object allows you to implement hook code for the SVN repository. The pre-commit and pre-revprop-change hooks are t only hooks that are currently appropriate in SVN. See the SVN documentation for details on hook scripts.

A Transaction object can only be used on one thread at a time. If two threads attempt to call methods of Transaction at the same time one of the threads will get a pysvn.TransactionError exception with the value 'transaction in use on another thread'.

When the optional parameter is_revision is True, than the transaction_name parameter will be interpreted as a revision number and all subsequent operation will be performed on this revision. Note that the propdel and propset operations will fail than. This option lets you use the Transation object to write post-commit hooks with the same API than pre-commit hooks, and lets you easily test your pre-comm hook on revisions.

pysvn.Client.vacuum

```
vacuum( path,
        remove_unversioned_items=False,
        remove_ignored_items=False,
        fix_recorded_timestamps=True,
        vacuum_pristines=True,
        include_externals=False )
```

Clean up any locks in the working copy at path. Usually such locks are the result of a failed or interrupted operation.

If remove_unversioned_items is True, remove unversioned items in path after successful working copy cleanup.

If remove_ignored_items is True, remove ignored unversioned items in path after successful working copy cleanup.

If fix_recorded_timestamps is True, this function fixes recorded timestamps for unmodified files in the working copy, reducing comparision time on future checks.

If vacuum_pristines is True, and dir_abspath points to the working copy root unreferenced files in the pristine store are removed.

If include_externals is True, recurse into externals and clean them up as well.

Transaction methods

cat	changed	list	
propdel	propget	proplist	propset
revpropdel	revpropget	revproplist	revpropset

pysvn.Transaction.cat

```
file_text = \
cat( path )
```

Return the contents of path as a string, file_text.

pysvn.Transaction.changed

```
file_text = \
changed( copy_info=False,
         base_dir="",
         low_water_mark=pysvn.Revision( opt_revision_kind.number, -1 ),
         send_deltas=False )
```

Return a dict of all changes in the transaction. The keys in the dict are the path names and the values are tuples containing the followir

- action - string - a single letter indicating the action 'A' for add, 'R' for modify, 'D' for delete
- kind - node_kind - one of the pysvn.node_kind values
- text_mod - int - is != 0 if the text in this path has been modified
- prop_mod - int - is != 0 if the properties in this path have been modified
- copyfrom_rev - int - when path has been copied this is its copy from revision number
- copyfrom_path - int - when path has been copied this is its copy from path

Changes will be limited to those within base_dir, and if low_water_mark is set to something other than SVN_INVALID_REVNUM(-1) it is assumed that the client has no knowledge of revisions prior to low_water_mark. Together, these two arguments define the portion of tl tree that the client is assumed to have knowledge of, and thus any copies of data from outside that part of the tree will be sent in their entirety, not as simple copies or deltas against a previous version.

The editor passed to this function should be aware of the fact that, if send_deltas is False, calls to its change_dir_prop(), change_file_prop(), and apply_textdelta() functions will not contain meaningful data, and merely serve as indications that properties or textual contents were changed.

If send_deltas is True, the text and property deltas for changes will be sent, otherwise null text deltas and empty prop changes will be used.

Note: This editor driver passes SVN_INVALID_REVNUM (-1) for all revision parameters in the editor interface except the copyfrom parameter of the add_file() and add_directory() editor functions.

pysvn.Transaction.list

```
path_content = list( [path] )
```

Return a dict of all entries in the directory 'path'. The keys in the dict are the path names and the value contains the kind (one of the pysvn.node_kind values). If 'path' is not given the root of the repository will be examined. This is a same as '' and '/' as path.

pysvn.Transaction.propdel

```
propdel( prop_name,
         path )
```

Delete the property prop_name from path in the transaction.

pysvn.Transaction.propget

```
prop_value = \
propget( prop_name,
         path )
```

Returns the prop_value as a string or None if the prop_name is not in the transaction.

pysvn.Transaction.proplist

```
prop_dict = \
proplist( path )
```

Returns a prop_dict. The prop_dict contains the prop_names and their values if set on the path in the transaction.

pysvn.Transaction.propset

```
propset( prop_name,
         prop_value,
         path )
```

Set the property prop_name to prop_value in path in the transaction.

pysvn.Transaction.revpropdel

```
revpropdel( prop_name )
```

Delete the revision property prop_name in the transaction.

pysvn.Transaction.revpropget

```
prop_val = \
revpropget( prop_name )
```

Returns the prop_val with the revision property value or None if not set in the transaction.

pysvn.Transaction.revproplist

```
prop_dict = \
revproplist()
```

Returns a prop_dict where the prop_dict contains the revision properies and their values in the transaction.

pysvn.Transaction.revpropset

```
rev = \
revpropset( prop_name,
            prop_value )
```

set the revision property prop_name to prop_value in path in the transaction. The revision updated is returned.

pysvn.Revision - subversion revision

The Revision object has three member variables:

- kind - the kind of revision, its value is one of the opt_revision_kind enumerations.
- date - date and time when kind is opt_revision_kind.date, as seconds since the epoch which is compatible with python's time modu
- number - revision number when kind is opt_revision_kind.number

Interface summary:

```
import pysvn
import time

revhead = pysvn.Revision( pysvn.opt_revision_kind.head )
revdate = pysvn.Revision( pysvn.opt_revision_kind.date, time.time() )
revnum = pysvn.Revision( pysvn.opt_revision_kind.number, 4721 )
```

pysvn.ClientError - Exception class raised by client commands on error

ClientError exception is raised when any of the subversion functions called by pysvn return an error.

The Client.exception_style variable controls the information stored in the ClientError object.

exception_style = 0

The args property is set to a single string parameter containing the whole error message. '\n' is used to seperate message parts.

Use str() to get the string description of the exception raised by pysvn.

```
import pysvn

client = pysvn.Client()
client.exception_style = 0
try:
```

```
    client.update( '.' )
except pysvn.ClientError, e:
    # convert to a string
    print str(e)
    # or access the string in args directly
    print e.args
```

exception_style = 1

The arg property is set to a tuple contain two values.

arg[0] is set to a string parameter containing the whole error message. '\n' is used to seperate message parts.

arg[1] is set to a list of tuples containing the message string and the error code. The error code values are defined by SVN and APR.

```
import pysvn

client = pysvn.Client()
client.exception_style = 1
try:

    client.update( '' )
except pysvn.ClientError, e:
    # print the whole message
    print e.args[0]
    # or process the error list
    for message, code in e.args[1]:
        print 'Code:',code,'Message:',message
```

pysvn.PysvnStatus - subversion status object

Each status object has the following fields:

- path - string - the path name
- entry - PysvnEntry - entry information
- is_versioned - Boolean - true if the path is versioned
- is_locked - Boolean - true if the path is locked
- is_copied - Boolean - true if the path is copied
- is_switched - Boolean - true if the path has been switched
- prop_status - wc_status_kind - the status of the properties of the path
- text_status - wc_status_kind - the status of the text of the path
- repos_prop_status - wc_status_kind - the repository status of the properties of the path
- repos_text_status - wc_status_kind - the repository status of the text of the path
- repos_lock - dict - the repository lock information

pysvn.PysvnEntry - subversion entry object

- checksum - string
- commit_author - string
- commit_revision - pysvn.Revision
- commit_time - time
- conflict_new - string or None - file path
- conflict_old - string of None - file path
- conflict_work - string of None - file path
- copy_from_revision - pysvn.Revision or None
- copy_from_url - string or None
- is_absent - boolean
- is_copied - boolean
- is_deleted - boolean
- is_valid - boolean
- kind - pysvn.node_kind
- name - string
- properties_time - time
- property_reject_file - string or None
- repos - string
- revision - pysvn.Revision or None
- schedule - pysvn.wc_schedule or None
- text_time - time
- url - string or None
- uuid - string or None

pysvn.opt_revision_kind - subversion revision number kind enumeration

- unspecified - No revision information given.
- number - revision given as number

- date - revision given as date
- committed - rev of most recent change
- previous - (rev of most recent change) - 1
- base - .svn/entries current revision
- working - current, plus local mods
- head - repository youngest

pysvn.wc_notify_action - subversion notification callback action enumeration

- add - Adding a path to revision control.
- copy - Copying a versioned path.
- delete - Deleting a versioned path.
- restore - Restoring a missing path from the pristine text-base.
- revert - Reverting a modified path.
- failed_revert - A revert operation has failed.
- resolved - Resolving a conflict.
- skip - Skipping a path.
- update_delete - Got a delete in an update.
- update_add - Got an add in an update.
- update_update - Got any other action in an update.
- update_completed - The last notification in an update (including updates of externals).
- update_external - Updating an external module.
- status_completed - The last notification in a status (including status on externals).
- status_external - Running status on an external module.
- commit_modified - Committing a modification.
- commit_added - Committing an addition.
- commit_deleted - Committing a deletion.
- commit_replaced - Committing a replacement.
- commit_postfix_txdelta - Transmitting post-fix text-delta data for a file.
- annotate_revision - Processed a single revision's blame.
- locked - Locking a path.
- unlocked - Unlocking a path.
- failed_lock - Failed to lock a path.
- failed_unlock - Failed to unlock a path.

pysvn.wc_status_kind - subversion status kind enumeration

- none - does not exist
- unversioned - is not a versioned thing in this wc
- normal - exists, but uninteresting.
- added - is scheduled for addition
- missing - under v.c., but is missing
- deleted - scheduled for deletion
- replaced - was deleted and then re-added
- modified - text or props have been modified
- merged - local mods received repos mods
- conflicted - local mods received conflicting repos mods
- ignored - a resource marked as ignored
- obstructed - an unversioned resource is in the way of the versioned resource
- external - an unversioned path populated by an svn:external property
- incomplete - a directory doesn't contain a complete entries list

pysvn.wc_merge_outcome - subversion merge outcome enumeration

- unchanged - The working copy is (or would be) unchanged. The changes to be merged were already present in the working copy
- merged - The working copy has been (or would be) changed.
- conflict - The working copy has been (or would be) changed, but there was (or would be) a conflict
- no_merge - No merge was performed, probably because the target file was either absent or not under version control.

pysvn.wc_notify_state - subversion notify callback state enumeration

- inapplicable - inapplicable
- unknown - Notifier doesn't know or isn't saying.
- unchanged - The state did not change.
- missing - The item wasn't present.
- obstructed - An unversioned item obstructed work.
- changed - Pristine state was modified.
- merged - Modified state had mods merged in.
- conflicted - Modified state got conflicting mods.

pysvn.wc_schedule - subversion status schedule enumeration

- normal - Nothing special here
- add - Slated for addition
- delete - Slated for deletion
- replace - Slated for replacement (delete + add)

pysvn.node_kind - subversion node kind enumeration

- none - absent
- file - regular file
- dir - directory
- unknown - something's here, but we don't know what

pysvn.depth - subversion depth enumeration

- empty - Just the named directory D, no entries. Updates will not pull in any files or subdirectories not already present.
- exclude - not used yet
- files - D + its file children, but not subdirs. Updates will pull in any files not already present, but not subdirectories.
- immediates - D + immediate children (D and its entries). Updates will pull in any files or subdirectories not already present; those subdirectories' this_dir entries will have depth-empty.
- infinity - D + all descendants (full recursion from D). Updates will pull in any files or subdirectories not already present; those subdirectories' this_dir entries will have depth-infinity. Equivalent to the pre-1.5 default update behavior.
- unknown - Depth undetermined or ignored