Watch a Directory for Changes

Introduction

The requirement: to know when files have been added, removed or updated within a directory.

There are several approaches you can take to monitoring a directory under Win32. I'm going to compare three: the first, to poll the directory every few seconds using os.listdir; the second, to use the FindFirstChangeNotification API; the third, to use the ReadDirectoryChanges API. Each is presented with its pros & cons. There are more sophisticated alternatives involving audit logging and volume change management, but I know next-to-nothing about them, and I suspect they're OTT for most people.

For a working module which provides a generator using the ReadDirectoryChanges approach, and an example of a threaded approach, fetch watch_directory.py (<2k)

Poll the directory with os.listdir

The approach is simple: every few seconds use os.listdir to pull up a list of the files in a directory. Compare this list against the previous run to see which files have been added or removed. For common-sense reasons, dictionaries are used to hold the list of files each time round.

- Easy to write and to understand
- Uses only standard modules
- Works across any platform
- Runs in respectable time for small-medium number of files

- Doesn't scale well (although it's not too bad if you don't expect many changes)
- Doesn't account for updated files (although you could get fancy with os.stat)

```
import os, time
path_to_watch = "."
before = dict ([(f, None) for f in os.listdir (path_to_watch)])
while 1:
  time.sleep (10)
  after = dict ([(f, None) for f in os.listdir (path_to_watch)])
  added = [f for f in after if not f in before]
  removed = [f for f in before if not f in after]
  if added: print "Added: ", ", ".join (added)
  if removed: print "Removed: ", ", ".join (removed)
  before = after
```

Use the FindFirstChangeNotification API

The approach here is to use the MS FindFirstChangeNotification API, exposed via the pywin32 win32file module. It needs a little explanation: you get a change handle for a directory (optionally with its subdirectories) for certain kinds of change. You then use the ubiquitous WaitForSingleObject call from win32event, which fires when something's changed in one of your directories. Having noticed that something's changed, you're back to os.listdir-scanning to compare the before and after images. Repeat to fade.

NB: Only call FindNextChangeNotification if the FindFirst... has fired, *not* if it has timed out.

- Notifies only when a directory actually changes, so no polling needed
- Finer-grained tuning on the change-notification: you could notify on a size change, for example

- The code is more complex to understand and manage
- You still have to compare the directory listings to find out what's changed

```
import os

import win32file
import win32event
import win32con

path_to_watch = os.path.abspath (".")

#
# FindFirstChangeNotification sets up a handle for watching
#  file changes. The first parameter is the path to be
#  watched; the second is a boolean indicating whether the
#  directories underneath the one specified are to be watched;
#  the third is a list of flags as to what kind of changes to
#  watch for. We're just looking at file additions / deletions.
#
change_handle = win32file.FindFirstChangeNotification (
  path_to_watch,
  0,
  win32con.FILE_NOTIFY_CHANGE_FILE_NAME
)
```

```
#
# Loop forever, listing any file changes. The WaitFor... will
#  time out every half a second allowing for keyboard interrupts
#  to terminate the loop.
#
try:

  old_path_contents = dict ([(f, None) for f in os.listdir (path_to_watch)])
  while 1:
    result = win32event.WaitForSingleObject (change_handle, 500)

    #
    # If the WaitFor... returned because of a notification (as
    #  opposed to timing out or some error) then look for the
    #  changes in the directory contents.
    #
    if result == win32con.WAIT_OBJECT_0:
      new_path_contents = dict ([(f, None) for f in os.listdir (path_to_watch)])
      added = [f for f in new_path_contents if not f in old_path_contents]
      deleted = [f for f in old_path_contents if not f in new_path_contents]
      if added: print "Added: ", ", ".join (added)
      if deleted: print "Deleted: ", ", ".join (deleted)

      old_path_contents = new_path_contents
      win32file.FindNextChangeNotification (change_handle)

finally:
  win32file.FindCloseChangeNotification (change_handle)
```

Use the ReadDirectoryChanges API

The third technique uses the MS ReadDirectoryChanges API, exposed via the pywin32 win32file module. The way we employ it here is to use call ReadDirectoryChangesW in blocking mode. Similarly to the FindFirstChange approach (but slightly differently — thank you, Microsoft!) we specify what changes are to be notified and whether or not to watch subtrees. Then you just wait... The function returns a list of 2-tuples, each one representing an action and a filename. A rename always gives a pair of 2-tuples; other compound actions may also give a list.
Obviously, you could get fancy with a micro state machine to give better output on renames and other multiple actions.

- Notifies only when a directory actually changes, so no polling needed
- Finer-grained tuning on the change-notification: you could notify on a size change, for example
- You know which files have been altered, and how (at least up to a point: you don't know *how* a file has been updated or by whom, merely that it has been).

- The code is harder to write and understand (especially the initial handle) than the naive approach, but is still easier than the FindFirstChange approach.
- Because it's a blocking call, it's difficult to get out of the loop if no change is made to the directory.

Update: Daniel D pointed out that, without FILE_SHARE_DELETE on the CreateFile call, the directory can't be deleted or renamed while it's being watched.

```
import os

import win32file
import win32con

ACTIONS = {
  1 : "Created",
  2 : "Deleted",
  3 : "Updated",
  4 : "Renamed from something",
  5 : "Renamed to something"
}
# Thanks to Claudio Grondi for the correct set of numbers
FILE_LIST_DIRECTORY = 0x0001

path_to_watch = "."
hDir = win32file.CreateFile (
  path_to_watch,
  FILE_LIST_DIRECTORY,
  win32con.FILE_SHARE_READ | win32con.FILE_SHARE_WRITE | win32con.FILE_SHARE_DELETE,
  None,
  win32con.OPEN_EXISTING,
  win32con.FILE_FLAG_BACKUP_SEMANTICS,
  None
)
while 1:
  #
  # ReadDirectoryChangesW takes a previously-created
  # handle to a directory, a buffer size for results,
  # a flag to indicate whether to watch subtrees and
  # a filter of what changes to notify.
  #
  # NB Tim Juchcinski reports that he needed to up
  # the buffer size to be sure of picking up all
  # events when a large number of files were
  # deleted at once.
  #
  results = win32file.ReadDirectoryChangesW (
```

```
    hDir,
    1024,
    True,
    win32con.FILE_NOTIFY_CHANGE_FILE_NAME |
     win32con.FILE_NOTIFY_CHANGE_DIR_NAME |
     win32con.FILE_NOTIFY_CHANGE_ATTRIBUTES |
     win32con.FILE_NOTIFY_CHANGE_SIZE |
     win32con.FILE_NOTIFY_CHANGE_LAST_WRITE |
     win32con.FILE_NOTIFY_CHANGE_SECURITY,
    None,
    None
  )
  for action, file in results:
    full_filename = os.path.join (path_to_watch, file)
    print full_filename, ACTIONS.get (action, "Unknown")
```