





Introducing Java 7

MOVING JAVA FORWARD



ORACLE®

The New File System API in JDK 7

Staffan Friberg

Agenda

- Background
- File System API
 - Common classes
 - API Features
 - Service Provider Interface
- Summary

Background and Motivation

- Need something better than `java.io.File`
 - Doesn't work consistently across platforms
 - No useful exceptions when a file operation fails
 - Missing basic operations (file copy, move, ...)
 - Limited support for symbolic links
 - Limited support for file attributes, performance issues
 - No way to plug-in other file system implementations
- Need a new file system API
 - JSR-203 tasked to define this and more

Main concepts

- Path
 - Locates a file in a file system
- Files
 - Static methods to operate on files and directories
- FileSystem
 - A handle to a file system and factory for objects that access it
 - `FileSystems.getDefault()` returns the local/platform file system
- FileStore
 - The underlying storage (volume, concrete file system ...)

Path

- Immutable
- Create from path String, URI or java.io.File
- Syntax is essentially [name]+ or root [name]*
 - foo, /, /foo, foo/bar C:\, C:\foo, \\foofighter\bar
- Defines methods to access, compare and modify Paths

- Support old libraries
 - Create File from Path using toFile

Files

- Static methods to operate on files and directories
 - Create/open for reading and write
 - Copy, move and delete
 - Read and set file attributes
 - ...
- Methods accessing the filesystem may throw IOException
 - Specific exceptions for some recoverable errors

File Attributes

- Groups of related attributes
- Define view that provides
 - Typesafe access to attributes in group
 - Bulk access where appropriate
 - Convert to/from the file system representation
- Implementations required to support Basic view
 - Common file systems attributes (size, type, timestamps)
- Implementations may support additional views

File Attributes

```
PosixFileAttributes attrs =  
    Files.readAttributes(path, PosixFileAttributes.class);  
  
UserPrincipal owner = attrs.owner();  
UserPrincipal group = attrs.group();  
Set<PosixFilePermission> perms = attrs.permissions();  
  
Files.createFile(newPath,  
    PosixFilePermissions.asFileAttribute(perms));
```

DirectoryStream

- DirectoryStream to iterate over entries
 - Scales to large directories and uses less resources
 - Smooth out response time for remote file systems
 - Provides handle to open directory
- Filtering
 - Built-in support for glob and regex patterns and custom filters
- DirectoryStream
 - Extends Iterable to allow use of for-each construct
 - Extends Closeable, need to remember to close stream

DirectoryStream

```
Path dir = ...
```

```
try (DirectoryStream<Path> stream =  
    Files.newDirectoryStream(dir, "*.java")) {  
    for (Path entry : stream) {  
        System.out.println(entry.getName());  
    }  
}
```

Symbolic Links

- Optionally supported
- API based on long standing Unix semantics
 - Windows Vista or newer with NTFS
- Symbolic links followed by default, some exceptions
 - delete, moveTo, walkFileTree
- Files.isSameFile
 - Check if two paths reference the same file

Recursive Operations

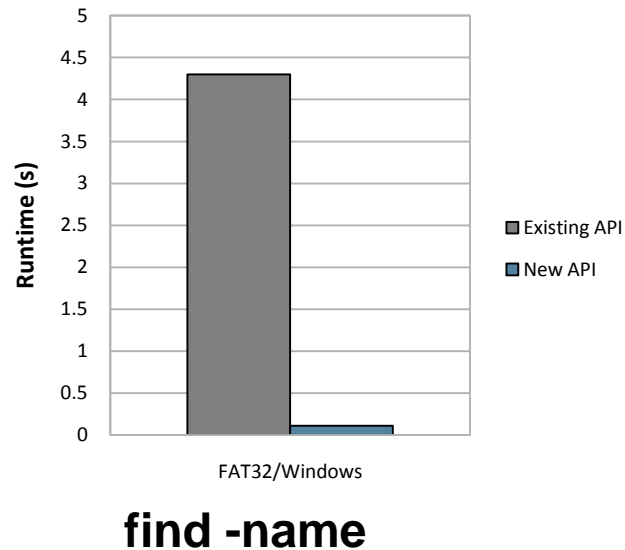
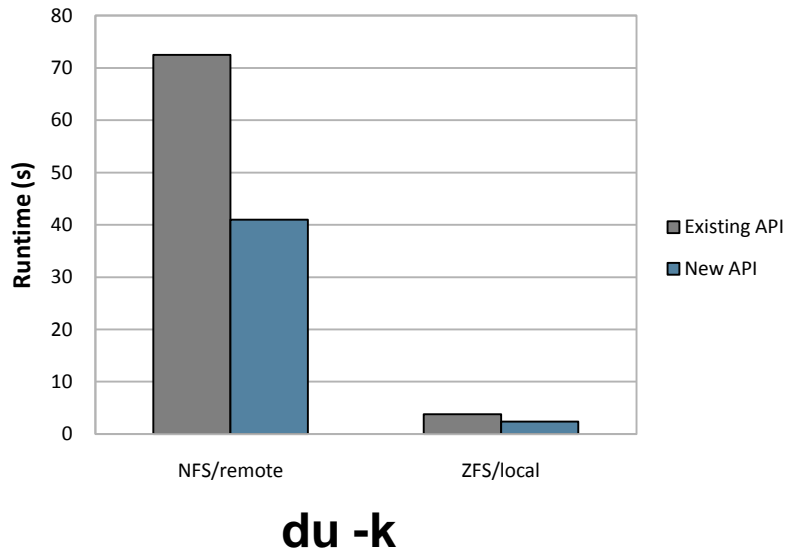
- Files.walkFileTree
 - Internal iterator to walk a file tree from a given starting point
 - FileVisitor invoked for each file/directory encountered
 - Depth first, invoked twice for each directory (pre/post)

```
interface FileVisitor<T> {  
    FileVisitResult preVisitDirectory(T dir, BasicFileAttributes attrs);  
    FileVisitResult visitFile(T file, BasicFileAttributes attrs);  
    FileVisitResult visitFileFailed(T file, IOException ioe);  
    FileVisitResult postVisitDirectory(T dir, IOException ioe);  
}
```

Recursive Operations

- Files.walkFileTree
 - Internal iterator to walk a file tree from a given starting point
 - FileVisitor invoked for each file/directory encountered
 - Depth first, invoked twice for each directory (pre/post)
 - Return value controls iteration, can also limit depth
 - Easy to develop recursive operations
 - SimpleFileVisitor with default behavior
- Symbolic links not followed by default, use FOLLOW_LINKS
 - Cycles detected and reported

Performance



File change notification

- Watch files and directories for changes
 - Typically done by polling
 - Scanning directories, checking file timestamps, ...
- WatchService
 - Watches registered objects for events
 - Makes use of native event notification facility where possible
 - All providers required to support monitoring of directories
 - Events when files are created, deleted, or modified
 - Extendable to other objects and events

File change notification – Registration

```
WatchService watcher =  
    FileSystems.getDefault().newWatchService();  
  
Path dir = ...  
WatchKey key = dir.register(watcher, ENTRY_CREATE);
```

File change notification – Retrieving events

```
for (;;) {  
    WatchKey key = watcher.take();  
    for (WatchEvent<?> event: key.pollEvents()) {  
        if (event.kind() == ENTRY_CREATE) {  
            Path name = (Path)event.context();  
            System.out.format("%s created%n", name);  
        }  
    }  
    key.reset();  
}
```

Service Provider Interface

- FileSystemProvider is the factory for FileSystem instances
- Develop and deploy custom file system implementations
- Deploy as JAR file on class path or in extensions directory

- ZIP file system
 - Example provider included in the JDK 7 demos
 - Treats contents of zip or JAR file as a file system

Summary

- The JDK gets a comprehensive interface to the file system
- Easy to use, yet powerful and extendable
- Code samples and demos available in JDK 7

