



JavaOne™

java.sun.com/javaone

New I/O in JDK™ 7

Alan Bateman
Sun Microsystems Inc.

Carl Quinn
Google Inc.

TS-5686



Learn about the new File System API,
Asynchronous I/O, and the many other
updates to the New I/O APIs

GOAL

Outline

- File System API
- Channels API
 - Updates to socket channel API
 - Asynchronous I/O
- Miscellaneous Topics
- Conclusion

Outline

- ***File System API***
- Channels API
 - Updates to socket channel API
 - Asynchronous I/O
- Miscellaneous
- Conclusion

What's wrong with `java.io.File`?

New File System API

➤ New packages:

- `java.nio.file`, `java.nio.file.attribute`

➤ Main classes:

- `FileSystem`
 - Interface to file system
 - Factory for objects to access files and other objects in file system
- `FileRef`
 - Reference to file or directory
 - Defines methods to operate on file or directory
- `Path`
 - A `FileRef` that locates a file by a system dependent path
 - Created by `FileSystem` by converting path string or URI
- `FileStore`
 - Underlying storage pool, device, partition...

Hello World

```
import java.nio.file.*;

Path home = Path.get("/home/gus");

Path profile = home.resolve(".bash_profile");

// Backup existing file
profile.copyTo(home.resolve(".bash_profile.backup"));

// Append useful stuff to the profile
WritableByteChannel ch = profile.newSeekableByteChannel(APPEND);
try {
    appendStuff(ch);
} finally {
    ch.close();
}
```

Overview of Path

- Methods to access components
- Methods to test and compare
- Methods to combine paths
- File operations
- All methods that access file system throw IOException
 - No other checked exceptions in API

Opening/Creating Files

➤ Stream I/O

- `newInputStream` and `newOutputStream` methods
- Interoperability with `java.io` package

➤ Channel I/O

- `java.nio.channels.SeekableByteChannel`
 - `ByteChannel` that maintains a position
 - Channel equivalent of `RandomAccessFile`
- `newSeekableByteChannel` methods to open or create file
- `InterruptibleChannel` semantics
 - Asynchronously closeable and interruptible
- `READ`, `WRITE`, `APPEND`, `TRUNCATE_EXISTING`, `CREATE`, `CREATE_NEW`, `NOFOLLOW_LINKS`, `SYNC`, `DSYNC`...

➤ Optionally set initial attributes when creating files

- important for file permissions

Copying and Moving Files

- copyTo method to copy file to target location
 - Options to replace existing file, copy file attributes...
- moveTo method to move file to target location
 - Option to replace existing file
 - Option to require operation to be atomic

```
import static java.nio.file.StandardCopyOption.*;  
  
Path source = Path.get("C:\\My Documents\\Stuff.odp");  
Path target = Path.get("D:\\Backup\\MyStuff.odp");  
  
source.copyTo(target);  
  
source.copyTo(target, REPLACE_EXISTING, COPY_ATTRIBUTES);
```

Symbolic Links (1/2)

➤ Unix semantics

- Follow links by default except for delete and moveTo
- File attribute views can be configured to follow links or not
- Methods to create link, read target, and test if file is a symbolic link
- Works on Windows Vista

➤ More API support

- isSameFile to test if two objects reference the same file
- copyTo option to copy link (default is to copy final target)
- toRealPath option to expand links or not
- walkFileTree method detects loops when following links

➤ Basic support for hard links

- createLink, linkCount

Symbolic Links (2/2)

```
Path file = Path.get("/usr/spool");  
// read file attributes of the link  
BasicFileAttributes attrs = Attributes  
    .readBasicFileAttributes(file, false);  
  
if (attrs.isSymbolicLink()) {  
    // read target of link  
    Path target = file.readSymbolicLink();  
  
    // check /usr/spool == /usr/spool/../var/spool  
    assert file.isSameFile(file.resolve(target));  
}
```

Directories (1/2)

➤ DirectoryStream

- To iterate over the entries in a directory
- Scales to large directories
- Optional filter to decide if entries should be accepted or filtered
- Built-in filters to match file names using glob or regular expression

```
Path dir = Path.get("mydir");

DirectoryStream stream = dir.newDirectoryStream("*.java");
try {
    for (DirectoryEntry entry: stream) {
        System.out.println(entry.getName());
    }
} finally {
    stream.close();
}
```

Directories (1/2)

➤ DirectoryStream

- To iterate over the entries in a directory
- Scales to large directories
- Optional filter to decide if entries should be accepted or filtered
- Built-in filters to match file names using glob or regular expression

```
Path dir = Path.get("mydir");

Files.withDirectory(dir, "*.java", new DirectoryAction() {
    public void invoke(DirectoryEntry entry) {
        System.out.println(entry.getName());
    }
});
```

Directories (2/2)

➤ DirectoryStreamFilters

- Factory methods for useful filters
- `newContentTypeFilter`
 - Accept entry based on its MIME type
 - Use installed file type detectors
- Combine filters into simple expressions

➤ `Files.walkFileTree` utility method

- Recursively descends directory hierarchy rooted at starting file
- Easy to use internal iterator
 - `FileVisitor` invoked for each directory/file in hierarchy
 - Options to control if symbolic links are followed, maximum depth...
- Use to implement recursive copy, move, delete, `chmod`...

File Attributes (1/4)

➤ Meta-data associated with files

- Time stamps, file owner, permissions...
- Highly platform/file system specific

➤ Approach:

- Organize related attributes into groups
- Define FileAttributeView that provides a view of these attributes
 - A view may extend or overlap with other views
 - May need to translate to/from file system representation
- BasicFileAttributeView provides a view of a basic set of attributes
 - required to be supported by all implementations
- Specification defines other FileAttributeView types
 - Access to POSIX, DOS, ACLs, Named...
- Implementation may support others

File Attributes (2/4)

- newFileAttributeView method
 - selects view by class literal that works as type-token
 - method returns instance of view

```
BasicFileAttributeView view =  
    file.newFileAttributeView(BasicFileAttributeView.class, true);  
  
// bulk read of basic attributes  
BasicFileAttributes attrs = view.readAttributes();
```

- Bulk read of BasicFileAttributes
 - size, isDirectory, isRegularFile, isSymbolicLink, lastModifiedTime, lastAccessTime...
- Attributes utility class makes it easy for common cases

File Attributes (2/4)

- newFileAttributeView method
 - selects view by class literal that works as type-token
 - method returns instance of view

```
// bulk read of basic attributes
BasicFileAttributes attrs =
    Attributes.readBasicFileAttributes(file, true);
```

- Bulk read of BasicFileAttributes
 - size, isDirectory, isRegularFile, isSymbolicLink, lastModifiedTime, lastAccessTime...
- Attributes utility class makes it easy for common cases

File Attributes (3/4)

➤ PosixFileAttributeView

- Unix equivalent of stat, chmod, chown...

```
PosixFileAttributes attrs =  
    Attributes.readPosixFileAttributes(file, true);  
  
String mode = PosixFilePermission.toString(attrs.permissions());  
System.out.format("%s %s %s", mode, attrs.owner(), attr.group());  
  
    rwxrw-r-- alanb java  
  
import static java.nio.file.attribute.PosixFilePermission.*;  
  
Attributes.setPosixFilePermissions(file,  
    OWNER_READ, OWNER_WRITE, GROUP_WRITE, OTHERS_READ);
```

File Attributes (4/4)

➤ DosFileAttributeView

- Provides access to legacy DOS attributes
- Implementable "server-side" on non-Windows platforms

➤ AclFileAttributeView

- Provides access to Access Control Lists (ACLs)
- Based on NFSv4 ACL model

➤ NamedAttributeView

- Provides access to attributes as name/value pairs
- Mappable to file systems that support named subfiles

File change notification (1/2)

- Address need of applications that need to detect changes or events caused by *non communicating entities*
 - IDEs, poll directory for WAR files to deploy...
- WatchService
 - Watches registered objects for changes
 - Make use of inotify, FEN... where available
 - Consumer threads poll watch service to retrieve events
 - Multiple threads can service events concurrently
 - Easy to build listener interface for graphical applications

File change notification (2/2)

```
WatchService watcher = FileSystems.getDefault().newWatchService();

Set<StandardWatchEventType> events =
    EnumSet.of(ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);
WatchKey key = dir.register(watcher, events);

for (;;) {
    // wait for key to be signalled
    key = watcher.take();

    // process events
    for (WatchEvent<?> ev: key.pollEvents()) {

        if (event.getType() == ENTRY_MODIFY) {
            :
        }
    }

    // reset key
    key.reset();
}
```

Interoperability

- java.io.File getFileRef method to
 - Fix problems without major re-writes
 - Make it easy to make use of new features

```
File source = ...  
File target = ...  
  
if (!source.renameTo(target)) {  
    System.err.println("Your guess is as good as mine");  
}
```

- java.util.Scanner and java.util.Formatter updated

Interoperability

- java.io.File getFileRef method to
 - Fix problems without major re-writes
 - Make it easy to make use of new features

```
File source = ...
File target = ...

try {
    source.getFileRef().moveTo(target.getFileRef());
} catch (IOException x) {
    System.err.println(x);
}
```

- java.util.Scanner and java.util.Formatter updated

Provider interface

➤ Allows for:

- Replacement of default file system provider
- Interposing on default system provider
- Development and deployment of custom file systems

➤ Custom file systems:

- Develop FileSystemProvider implementation
- Factory for FileSystem objects
- FileSystem identified by URI
- Each FileSystem is distinct (no direct support for federation)
- Deploy as Java™ Archive (JAR) file as extension or use custom class loader
- Potential to load providers from repositories or module class loader when Java™ Module System integrated

Outline

- File System API
- Channels API
 - *Updates to socket channel API*
 - Asynchronous I/O
- Miscellaneous Topics
- Conclusion

Updates to Socket channel API (1/2)

➤ Motivation

- Network channels not complete abstraction of network socket
- Forced to mix channel and socket APIs to
 - bind, manipulate socket options...
- Legacy Socket behavior must be emulated by socket adapter
- Can't make use of platform specific socket options

➤ Approach

- NetworkChannel - channel to network socket
- Defines bind, getLocalAddress, setOption, getOption, methods
- Existing channel retrofitted to implement interface

Updates to Socket channel API (2/2)

➤ Multicasting

- MulticastChannel
 - A NetworkChannel that can join multicast groups
- Implemented by
 - DatagramChannel
 - AsynchronousDatagramChannel
- Use opportunity to bring platform support up to date
 - Source-specific multicasting (IGMPv3, MLDv2)

Outline

- File System API
- Channels API
 - Updates to socket channel API
 - *Asynchronous I/O*
- Miscellaneous Topics
- Conclusion

Asynchronous I/O

➤ Goal

- Asynchronous I/O API to both sockets and files
- Take advantage of operating system I/O facilities where available

➤ API

- Future style
 - Initiate I/O operation, returning `java.util.concurrent.Future`
 - Future interface defines methods to test or wait for completion
- Callback style
 - Specify `CompletionHandler` when invoking I/O operation
 - `CompletionHandler` invoked when I/O operation completes (or fails)

Asynchronous I/O: Future style

```
AsynchronousSocketChannel ch = AsynchronousSocketChannel.open();

// initiate connection
// wait for connection to be established or failure
Future<Void> result = ch.connect(remote);
result.get();

ByteBuffer buf = ...

// initiate read
Future<Integer> result = ch.read(buf);

// do something

// wait for read to complete
try {
    int bytesRead = result.get();
    :
} catch (ExecutionException x) {
    // failed
}
```

Asynchronous I/O: Callback-style

```
ByteBuffer buf = ...

// CompletionHandler invoked when read completes
ch.read(buffer, ..., new CompletionHandler<Integer,Void>() {

    public void completed(CompletableFuture<Integer,Void> result) {
        try {
            int bytesRead = result.getNow();

        } catch (IOException x) {
            // error handling
        }
    }

});
```


What about Threads?

➤ Who invokes the completion handler?

- Initiating thread
- Thread in channel group's thread pool

➤ Channel Group

- Encapsulates mechanics required to handle I/O completion
- Has associated thread pool
- Each asynchronous channel is bound to a group
 - default group
 - or specify group when creating channel
- Configured by parameters
 - ThreadFactory
 - maximum threads to handle I/O events
 - ...

Other Asynchronous I/O topics

- Timeouts
- Asynchronous close
- Cancellation
- Provider interface
 - For the adventurous

Outline

- File System API
- Channels API
 - Updates to socket channel API
 - Asynchronous I/O
- *Miscellaneous Updates*
- Conclusion

Miscellaneous Updates

- Infiniband (IB) Sockets Direct Protocol (SDP)
 - Standard wire protocol over IB for stream oriented sockets
 - Uses Internet Protocol addressing
 - Use existing networking or socket-channel API
 - Select SDP protocol when creating socket or channel
- Stream Control Transport Protocol (SCTP)
 - Support blocking and non-blocking modes
 - Notifications when association changes, send fails, shutdown...
 - SctpSocketChannel
 - One-to-one style, similar to TCP, one SCTP association
 - SctpOneToManySocketChannel
 - One-to-many style, similar to UDP, multiple SCTP associations, association branch-off...

Outline

- File System API
- Channels API
 - Updates to socket channel API
 - Asynchronous I/O
- Miscellaneous Topics
- *Conclusion*

More information

- JSR page:
 - <http://jcp.org/en/jsr/detail?id=203>
- Project page:
 - <http://openjdk.java.net/projects/nio/>
 - Source code now
- Mailing lists
 - nio-dev@openjdk.java.net
 - nio-discuss@openjdk.java.net
- Coming soon
 - Early access binaries
 - More samples

THANK YOU



Alan Bateman

Sun Microsystems Inc.

Carl Quinn

Google Inc.

TS-5686

