Module java.base

Package java.nio.file

`package java.nio.file`

Defines interfaces and classes for the Java virtual machine to access files, file attributes, and file systems.

The java.nio.file package defines classes to access files and file systems. The API to access file and file system attributes is defined in the `java.nio.file.attribute` package. The `java.nio.file.spi` package is used by service provider implementors wishing to extend the platform default provider, or to construct other provider implementations.

Symbolic Links

Many operating systems and file systems have support for *symbolic links*. A symbolic link is a special file that serves as a reference to another file. For the most part, symbolic links are transparent to applications and operations on symbolic links are automatically redirected to the *target* of the link. Exceptions to this are when a symbolic link is deleted or renamed/moved in which case the link is deleted or removed rather than the target of the link. This package includes support for symbolic links where implementations provide these semantics. File systems may support other types that are semantically close but support for these other types of links is not included in this package.

Interoperability

The `File` class defines the `toPath` method to construct a `Path` by converting the abstract path represented by the `java.io.File` object. The resulting `Path` can be used to operate on the same file as the `File` object. The `Path` specification provides further information on the interoperability between `Path` and `java.io.File` objects.

Visibility

The view of the files and file system provided by classes in this package are guaranteed to be consistent with other views provided by other instances in the same Java virtual machine. The view may or may not, however, be consistent with the view of the file system as seen by other concurrently running programs due to caching performed by the underlying operating system and delays induced by network-filesystem protocols. This is true regardless of the language in which these other programs are written, and whether they are running on the same machine or on some other machine. The exact nature of any such inconsistencies is system-dependent and therefore unspecified.

Synchronized I/O File Integrity

The `SYNC` and `DSYNC` options are used when opening a file to require that updates to the file are written synchronously to the underlying storage device. In the case of the default provider, and the file resides on a local storage device, and the `seekable` channel is connected to a file that was opened with one of these options, then an invocation of the `write` method is only guaranteed to return when all changes made to the file by that invocation have been written to the device. These options are useful for ensuring that critical information is not lost in the event of a system crash. If the file does not reside on a local device then no such guarantee is made. Whether this guarantee is possible with other `provider` implementations is provider specific.

General Exceptions

Unless otherwise noted, passing a `null` argument to a constructor or method of any class or interface in this package will cause a `NullPointerException` to be thrown. Additionally, invoking a method with an array or collection containing a `null` element will cause a `NullPointerException`, unless otherwise specified.

Unless otherwise noted, methods that attempt to access the file system will throw `ClosedFileSystemException` when invoked on objects associated with a `FileSystem` that has been `closed`. Additionally, any methods that attempt write access to a file system will throw `ReadOnlyFileSystemException` when invoked on an object associated with a `FileSystem` that only provides read-only access.

Unless otherwise noted, invoking a method of any class or interface in this package created by one `provider` with a parameter that is an object created by another provider, will throw `ProviderMismatchException`.

Optional Specific Exceptions

Most of the methods defined by classes in this package that access the file system specify that `IOException` be thrown when an I/O error occurs. In some cases, these methods define specific I/O exceptions for common cases. These exceptions, noted as *optional specific exceptions*, are thrown by the implementation where it can detect the specific error. Where the specific error cannot be detected then the more general `IOException` is thrown.

Since:

1.7

Related Packages

Package	Description
java.nio	Defines buffers, which are containers for data, and provides an overview of the other NIO packages.
java.nio.file.attribute	Interfaces and classes providing access to file and file system attributes.
java.nio.file.spi	Service-provider classes for the `java.nio.file` package.
java.nio.channels	Defines channels, which represent connections to entities that are capable of performing I/O operations, such as files and sockets; defines selectors, for multiplexed, non-blocking I/O operations.

java.nio.charset	Defines charsets, decoders, and encoders, for translating between bytes and Unicode characters.

All Classes and Interfaces | **Interfaces** | **Classes** | **Enum Classes** | **Exception Classes**

Class	Description
AccessDeniedException	Checked exception thrown when a file system operation is denied, typically due to a file permission or other access check.
AccessMode	Defines access modes used to test the accessibility of a file.
AtomicMoveNotSupportedException	Checked exception thrown when a file cannot be moved as an atomic file system operation.
ClosedDirectoryStreamException	Unchecked exception thrown when an attempt is made to invoke an operation on a directory stream that is closed.
ClosedFileSystemException	Unchecked exception thrown when an attempt is made to invoke an operation on a file and the file system is closed.
ClosedWatchServiceException	Unchecked exception thrown when an attempt is made to invoke an operation on a watch service that is closed.
CopyOption	An object that configures how to copy or move a file.
DirectoryIteratorException	Runtime exception thrown if an I/O error is encountered when iterating over the entries in a directory.
DirectoryNotEmptyException	Checked exception thrown when a file system operation fails because a directory is not empty.
DirectoryStream<T>	An object to iterate over the entries in a directory.
DirectoryStream.Filter<T>	An interface that is implemented by objects that decide if a directory entry should be accepted or filtered.
FileAlreadyExistsException	Checked exception thrown when an attempt is made to create a file or directory and a file of that name already exists.

Files	This class consists exclusively of static methods that operate on files, directories, or other types of files.
FileStore	Storage for files.
FileSystem	Provides an interface to a file system and is the factory for objects to access files and other objects in the file system.
FileSystemAlreadyExistsException	Runtime exception thrown when an attempt is made to create a file system that already exists.
FileSystemException	Thrown when a file system operation fails on one or two files.
FileSystemLoopException	Checked exception thrown when a file system loop, or cycle, is encountered.
FileSystemNotFoundException	Runtime exception thrown when a file system cannot be found.
FileSystems	Factory methods for file systems.
FileVisitOption	Defines the file tree traversal options.
FileVisitor<T>	A visitor of files.
FileVisitResult	The result type of a `FileVisitor`.
InvalidPathException	Unchecked exception thrown when path string cannot be converted into a `Path` because the path string contains invalid characters, or the path string is invalid for other file system specific reasons.
LinkOption	Defines the options as to how symbolic links are handled.
LinkPermission	The `Permission` class for link creation operations.
NoSuchFileException	Checked exception thrown when an attempt is made to access a file that does not exist.
NotDirectoryException	Checked exception thrown when a file system operation, intended for a directory, fails because the file is not a directory.
NotLinkException	Checked exception thrown when a file system operation fails because a file is not a symbolic link.

OpenOption	An object that configures how to open or create a file.
Path	An object that may be used to locate a file in a file system.
PathMatcher	An interface that is implemented by objects that perform match operations on paths.
Paths	This class consists exclusively of static methods that return a `Path` by converting a path string or `URI`.
ProviderMismatchException	Unchecked exception thrown when an attempt is made to invoke a method on an object created by one file system provider with a parameter created by a different file system provider.
ProviderNotFoundException	Runtime exception thrown when a provider of the required type cannot be found.
ReadOnlyFileSystemException	Unchecked exception thrown when an attempt is made to update an object associated with a `read-only` `FileSystem`.
SecureDirectoryStream<T>	A `DirectoryStream` that defines operations on files that are located relative to an open directory.
SimpleFileVisitor<T>	A simple visitor of files with default behavior to visit all files and to re-throw I/O errors.
StandardCopyOption	Defines the standard copy options.
StandardOpenOption	Defines the standard open options.
StandardWatchEventKinds	Defines the *standard* event kinds.
Watchable	An object that may be registered with a watch service so that it can be *watched* for changes and events.
WatchEvent<T>	An event or a repeated event for an object that is registered with a `WatchService`.
WatchEvent.Kind<T>	An event kind, for the purposes of identification.
WatchEvent.Modifier	An event modifier that qualifies how a `Watchable` is registered with a `WatchService`.
WatchKey	A token representing the registration of a `watchable` object with a `WatchService`.

WatchService	A watch service that *watches* registered objects for changes and events.