# smb-kotlin Developer Guide

# smb-kotlin Developer Guide

**Version 1.3.0** | Coconut Tree Software, Inc.

## Introduction

smb-kotlin is a pure Kotlin implementation of the SMB2 and SMB3 network file sharing protocols, packaged as a library for use in Kotlin and Java applications. It provides a modern, coroutine-based API for connecting to SMB shares and performing file operations including listing directories, creating directories, reading and writing files, copying, renaming, and deleting. The library supports NTLMv2 authentication, message signing, and AES-128-CCM encryption. It targets both JVM server applications and Android.

## Installation

### Maven Central (Licensed Build)

The licensed build is published on Maven Central. A valid license file is required at runtime.

#### Gradle (Kotlin DSL)

```
repositories {
    mavenCentral()
}

dependencies {
    implementation("com.ctreesoft:smb-kotlin:1.3.0")
}
```

**Gradle (Groovy DSL)**

```
repositories {
    mavenCentral()
}

dependencies {
    implementation 'com.ctreesoft:smb-kotlin:1.3.0'
}
```

### Runtime Dependencies

The library pulls in the following transitive dependencies automatically:

| Dependency | Purpose |
|---|---|
| `kotlinx-coroutines-core` | Coroutine support for async operations |
| `okio` | Streaming I/O for file read and write |
| `ktor-network` | Default NIO-based TCP transport |
| `bcprov-jdk15on` (Bouncy Castle) | Cryptography (NTLM, signing, encryption) |
| `slf4j-api` (1.7.x) | Logging facade |

You may need to add a logging backend depending on your platform. See the Logging section.

---

## Licensing

The Maven Central build requires a valid license to operate. Without a license, `SmbClient.connect()` throws `SmbLicenseException("License required")`.

### License File Format

Licenses are delivered as `.lic` text files:

```
-----BEGIN CTS LICENSE-----
Product: smb-kotlin
Licensee: Acme Corp
Expires: 2027-03-13
Signature: a3Fj8xR2...base64...7hGkQ=
-----END CTS LICENSE-----
```

Licenses are validated offline using Ed25519 digital signatures. No internet connectivity is required for validation.

## Loading a License

The `SmbLicense` class provides three factory methods for loading license files:

### From a File (JVM)

```kotlin
import com.ctreesoft.smb.api.SmbLicense
import java.io.File

val license = SmbLicense.fromFile(File("/etc/myapp/smb.lic"))
```

### From a String

```kotlin
val licenseText = """
    -----BEGIN CTS LICENSE-----
    Product: smb-kotlin
    Licensee: Acme Corp
    Expires: 2027-03-13
    Signature: a3Fj8xR2...base64...7hGkQ=
    -----END CTS LICENSE-----
""".trimIndent()

val license = SmbLicense.fromString(licenseText)
```

#### From an InputStream (Android Assets or Resources)

```
// From a classpath resource (JVM or Android)
val license = SmbLicense.fromStream(
    MyApp::class.java.getResourceAsStream("/smb.lic")!!
)

// From Android assets
val license = SmbLicense.fromStream(
    context.assets.open("smb.lic")
)
```

### Recommended Setup: Embed in Resources

The recommended approach is to place the `.lic` file in your project's resources so it ships inside the jar or APK.

**JVM:** Place the file at `src/main/resources/smb.lic` and load it:

```
val license = SmbLicense.fromStream(
    MyApp::class.java.getResourceAsStream("/smb.lic")!!
)
```

**Android:** Place the file at `app/src/main/assets/smb.lic` and load it:

```
val license = SmbLicense.fromStream(
    context.assets.open("smb.lic")
)
```

### Passing the License to SmbConfig

Once loaded, pass the license through `SmbConfig`:

```
val config = SmbConfig(license = license)
val client = SmbClient.connect("server", credentials, config = config)
```

### License Error Cases

| Scenario | Exception |
|---|---|
| No license provided | `SmbLicenseException("License required")` |
| Invalid or tampered license | `SmbLicenseException("Invalid license")` |
| Expired license | `SmbLicenseException("License expired on 2027-03-13")` |
| Valid license | No error, connection proceeds normally |

---

## Quick Start

### JVM / Server

```
import com.ctreesoft.smb.api.*
import kotlinx.coroutines.flow.toList
import kotlinx.coroutines.runBlocking
import java.io.File

fun main() {
    val license = SmbLicense.fromFile(File("smb.lic"))
    val credentials = NtlmCredentials("alice", "secret")
    val config = SmbConfig(license = license)

    runBlocking {
        SmbClient.connect("fileserver.local", credentials, config =
        config).use { client ->
            client.connectShare("documents").use { share ->
                val files = share.listDirectory("/").toList()
                files.forEach { println("${it.name} (${it.size} bytes)") }
            }
        }
    }
}
```

### Android

```
import com.ctreesoft.smb.api.*
import kotlinx.coroutines.flow.toList
import kotlinx.coroutines.launch
import kotlinx.coroutines.Dispatchers

class FileListViewModel(private val app: Application) :
        AndroidViewModel(app) {

    fun loadFiles(host: String, user: String, pass: String, shareName:
        String) {
        viewModelScope.launch(Dispatchers.IO) {
            val license = SmbLicense.fromStream(app.assets.open("smb.lic"))
            val credentials = NtlmCredentials(user, pass)
            val config = SmbConfig(license = license)

            SmbClient.connect(host, credentials, config = config).use {
        client ->
                client.connectShare(shareName).use { share ->
                    val entries = share.listDirectory("/").toList()
                    // Post entries to LiveData or StateFlow for UI
                }
            }
        }
    }
}
```

---

## API Reference

### SmbClient

The entry point for all SMB operations. Manages the TCP connection, SMB2 negotiation, and authentication session.

### Connecting

```
suspend fun SmbClient.Companion.connect(
    host: String,
    credentials: SmbCredentials,
    port: Int = 445,
    config: SmbConfig = SmbConfig.DEFAULT
): SmbClient
```

Establishes a TCP connection, negotiates SMB2/3, and authenticates using the provided credentials. Returns an authenticated `SmbClient`.

**Parameters:**

| Parameter | Type | Default | Description |
|---|---|---|---|
| `host` | `String` | — | Hostname or IP address of the SMB server |
| `credentials` | `SmbCredentials` | — | Authentication credentials |
| `port` | `Int` | `445` | TCP port number |
| `config` | `SmbConfig` | `SmbConfig.DEFAULT` | Client configuration |

**Throws:**

| Exception | When |
|---|---|
| `SmbLicenseException` | License is missing, invalid, or expired (licensed build only) |
| `SmbConnectionException` | Server is unreachable, connection refused, or DNS resolution fails |
| `SmbAuthenticationException` | Credentials are rejected by the server |

**Example:**

```
val client = SmbClient.connect(
    host = "192.168.1.100",
    credentials = NtlmCredentials("alice", "secret"),
    port = 445,
    config = SmbConfig(license = license)
)
```

### Accessing Shares

```
suspend fun connectShare(shareName: String): SmbShare
```

Connects to a named file share on the server. Pass the bare share name without UNC prefixes or backslashes.

**Throws:**

| Exception | When |
|---|---|
| `SmbShareNotFoundException` | The share does not exist on the server |
| `IllegalStateException` | The client is already closed |

**Example:**

```
val share = client.connectShare("documents")
```

You can open multiple shares simultaneously on the same client.

### Closing

```
fun close()
```

Closes all tracked shares and the underlying TCP session. Implements `Closeable`, so the idiomatic pattern is:

```
SmbClient.connect("server", credentials, config = config).use { client ->
    // work with client
}
```

Calling `close()` multiple times is safe.

---

## NtlmCredentials

NTLMv2 authentication credentials.

```
data class NtlmCredentials(
    val username: String,
    val password: String,
    val domain: String = ""
) : SmbCredentials
```

| Field | Type | Default | Description |
|---|---|---|---|
| `username` | `String` | — | User name |
| `password` | `String` | — | User password |
| `domain` | `String` | `""` | Domain or workgroup. Empty string uses the server's default domain. |

**Example:**

```
// Simple credentials (server's default domain)
val creds = NtlmCredentials("alice", "secret")

// With explicit domain
val creds = NtlmCredentials("alice", "secret", domain = "CORP")
```

---

## SmbConfig

Configuration for client behavior.

```
data class SmbConfig(
    val transferChunkSize: TransferChunkSize = TransferChunkSize.Default,
    val forceEncryption: Boolean = false,
    val socketFactory: SmbSocketFactory? = null,
    val resolveAddresses: Boolean = false,
    val license: SmbLicense? = null
)
```

| Field | Type | Default | Description |
|---|---|---|---|
| `transferChunkSize` | `TransferChunkSize` | `Default` | Controls READ/WRITE request sizes |
| `forceEncryption` | `Boolean` | `false` | Force AES-128-CCM encryption on all messages |
| `socketFactory` | `SmbSocketFactory?` | `null` | Custom socket factory for platform-specific transports |
| `resolveAddresses` | `Boolean` | `false` | Whether to resolve hostnames before connecting |
| `license` | `SmbLicense?` | `null` | License for authorized use (required on licensed builds) |

### TransferChunkSize

Controls the size of individual READ and WRITE SMB2 requests. Three modes are available:

```
// Default: 64KB chunks (industry standard, safest choice)
val config = SmbConfig(transferChunkSize = TransferChunkSize.Default)

// Server max: uses server's advertised maximum (typically up to 8MB)
// Best throughput for large file transfers
val config = SmbConfig(transferChunkSize = TransferChunkSize.ServerMax)

// Custom: specify an exact size in bytes
val config = SmbConfig(transferChunkSize =
        TransferChunkSize.Custom(262_144)) // 256KB
```

#### Forced Encryption

When `forceEncryption` is `true`, all messages are encrypted with AES-128-CCM regardless of whether the server requires it. The server must support SMB 3.x. Some servers require encryption through Group Policy settings. In that case, encryption is enabled automatically and this flag is not needed.

```
val config = SmbConfig(forceEncryption = true, license = license)
```

---

### SmbSocketFactory

A functional interface for providing custom `java.net.Socket` implementations. This is primarily used in enterprise environments like BlackBerry Dynamics where all network traffic must route through a secure tunnel.

```
fun interface SmbSocketFactory {
    fun createSocket(): Socket
}
```

When a `socketFactory` is provided, the library uses a blocking socket transport instead of the default ktor-network NIO transport. Socket I/O is wrapped in `Dispatchers.IO` for coroutine compatibility.

DNS resolution is deferred to the socket implementation using `InetSocketAddress.createUnresolved()` for hostname connections. This is required for environments where DNS must go through the secure tunnel.

**BlackBerry Dynamics Example:**

```
import com.good.gd.net.GDSocket

val config = SmbConfig(
    socketFactory = SmbSocketFactory { GDSocket() },
    license = license
)
val client = SmbClient.connect("internal-server", credentials, config =
        config)
```

**Standard Socket (for testing):**

```
import java.net.Socket

val config = SmbConfig(
    socketFactory = SmbSocketFactory { Socket() },
    license = license
)
```

---

## SmbShare

Represents a connection to an SMB file share. Provides all file operations. All paths are relative to the share root and may use forward slashes, which are internally normalized to backslashes. All operations are thread-safe and can be called from multiple coroutines concurrently.

---

### listDirectory

```
fun listDirectory(path: String): Flow<SmbFileEntry>
```

Returns a cold `Flow` that emits one `SmbFileEntry` for each file or subdirectory. The `.` and `..` entries are automatically filtered out. The underlying SMB2 exchange only occurs when the Flow is collected.

**Parameters:**

| Parameter | Type | Description |
|---|---|---|
| `path` | `String` | Directory path relative to share root. Empty string or `"/"` for the root. |

**Throws:**

| Exception | When |
|---|---|
| `SmbFileNotFoundException` | The directory does not exist |
| `SmbAccessDeniedException` | Access is denied |

**Examples:**

```
// Stream entries one at a time
share.listDirectory("reports/2024").collect { entry ->
    val type = if (entry.isDirectory) "DIR " else "FILE"
    println("$type ${entry.name} (${entry.size} bytes)")
}

// Collect all entries into a list
val entries = share.listDirectory("/").toList()

// Filter for specific files
val pdfs = share.listDirectory("documents")
    .filter { !it.isDirectory && it.name.endsWith(".pdf") }
    .toList()

// Count entries
val count = share.listDirectory("uploads").count()
```

---

### createDirectory

```
suspend fun createDirectory(path: String)
```

Creates a directory and all necessary parent directories, equivalent to `mkdir -p`. Succeeds silently if the directory already exists.

**Throws:**

| Exception | When |
|---|---|
| `SmbAccessDeniedException` | Access is denied |

**Example:**

```
// Creates "backups", "backups/2024", and "backups/2024/march" as needed
share.createDirectory("backups/2024/march")
```

---

### readFile

```
suspend fun <T> readFile(path: String, block: suspend (Source) -> T): T
```

Opens a file for reading, provides a streaming Okio `Source` to the lambda, and closes the file handle when the lambda returns or throws. The return value of the lambda is passed through.

Data is streamed in chunks controlled by `SmbConfig.transferChunkSize`, so arbitrarily large files can be read without loading the entire file into memory.

**Throws:**

| Exception | When |
|---|---|
| `SmbFileNotFoundException` | The file does not exist |
| `SmbAccessDeniedException` | Access is denied |

**Examples:**

```
// Read text content
val content = share.readFile("config/settings.json") { source ->
    source.buffer().readUtf8()
}

// Read binary data
val bytes = share.readFile("images/photo.jpg") { source ->
    source.buffer().readByteArray()
}

// Save remote file to local disk
share.readFile("data/export.csv") { source ->
    File("local-copy.csv").sink().buffer().use { localSink ->
        localSink.writeAll(source)
    }
}

// Android: save to app-specific storage
share.readFile("photos/image.jpg") { source ->
    val outFile = File(context.filesDir, "image.jpg")
    outFile.sink().buffer().use { localSink ->
        localSink.writeAll(source)
    }
}
```

---

**writeFile**

```
suspend fun writeFile(path: String, block: suspend (Sink) -> Unit)
```

Opens a file for writing, provides an Okio `Sink` to the lambda, and closes the file handle when the lambda returns or throws. If the file exists, it is overwritten. If it does not exist, it is created.

Data is sent in chunks, so there is no file size limit beyond available disk space on the server.

**Throws:**

| Exception | When |
| --- | --- |
| `SmbAccessDeniedException` | Access is denied |
| `SmbDiskFullException` | The server's disk is full |

**Examples:**

```
// Write text
share.writeFile("logs/output.txt") { sink ->
    sink.buffer().use { buf ->
        buf.writeUtf8("Hello from smb-kotlin!\n")
    }
}

// Upload a local file
share.writeFile("reports/report.pdf") { sink ->
    sink.buffer().use { buf ->
        buf.write(File("report.pdf").readBytes())
    }
}

// Stream a large file without loading into memory
share.writeFile("backup/database.dump") { sink ->
    val bufferedSink = sink.buffer()
    File("database.dump").source().buffer().use { localSource ->
        bufferedSink.writeAll(localSource)
    }
    bufferedSink.flush()
}

// Android: upload from content URI
share.writeFile("uploads/document.pdf") { sink ->
    val bufferedSink = sink.buffer()
    context.contentResolver.openInputStream(uri)!!.use { input ->
        bufferedSink.writeAll(input.source())
    }
    bufferedSink.flush()
}
```

---

### copyFile

```
suspend fun copyFile(sourcePath: String, destinationPath: String)
```

Copies a file on the server using server-side copy. The file data never transits through the client, making this operation fast regardless of file size. If the destination file exists, it is overwritten.

**Throws:**

| Exception | When |
|---|---|
| `SmbFileNotFoundException` | The source file does not exist |

**Example:**

```
share.copyFile("templates/invoice.docx", "invoices/2024-001.docx")
```

---

### rename

```
suspend fun rename(sourcePath: String, destinationPath: String)
```

Renames or moves a file or directory within the same share.

**Throws:**

| Exception | When |
|---|---|
| `SmbFileNotFoundException` | The source does not exist |
| `SmbFileExistsException` | The destination already exists |

**Example:**

```
share.rename("drafts/proposal.docx", "final/proposal-v2.docx")
```

---

### delete

```
suspend fun delete(path: String)
```

Deletes a single file or empty directory.

**Throws:**

| Exception | When |
|---|---|
| `SmbFileNotFoundException` | The path does not exist |
| `SmbDirectoryNotEmptyException` | The directory is not empty (use `deleteRecursively` instead) |
| `SmbAccessDeniedException` | Access is denied |

**Example:**

```
share.delete("temp/scratch.txt")
```

---

### deleteRecursively

```
suspend fun deleteRecursively(path: String)
```

Recursively deletes a directory and all its contents using depth-first traversal. All files and subdirectories are deleted before the directory itself.

**Throws:**

| Exception | When |
|---|---|
| `SmbFileNotFoundException` | The path does not exist |
| `SmbAccessDeniedException` | Access is denied to any entry |

**Example:**

```
share.deleteRecursively("temp/build-output")
```

---

#### close

```
fun close()
```

Closes the share connection. Implements `Closeable`:

```
client.connectShare("documents").use { share ->
    // file operations
}
```

---

### SmbFileEntry

Represents a file or directory entry returned by `listDirectory`.

```
data class SmbFileEntry(
    val name: String,
    val size: Long,
    val isDirectory: Boolean,
    val isReadOnly: Boolean,
    val isHidden: Boolean,
    val creationTime: Instant,
    val lastModifiedTime: Instant,
    val lastAccessTime: Instant
)
```

| Field | Type | Description |
|---|---|---|
| `name` | `String` | File or directory name (not a full path) |
| `size` | `Long` | File size in bytes (zero for directories) |
| `isDirectory` | `Boolean` | `true` if the entry is a directory |
| `isReadOnly` | `Boolean` | `true` if the read-only attribute is set |
| `isHidden` | `Boolean` | `true` if the hidden attribute is set |
| `creationTime` | `Instant` | When the file was created |
| `lastModifiedTime` | `Instant` | When the file was last written to |
| `lastAccessTime` | `Instant` | When the file was last accessed |

Timestamps are `java.time.Instant` values converted from Windows FILETIME format.

**Example:**

```
share.listDirectory("documents").collect { entry ->
    println(buildString {
        append(if (entry.isDirectory) "[DIR] " else "      ")
        append(entry.name.padEnd(30))
        append("${entry.size} bytes".padStart(15))
        append("  modified: ${entry.lastModifiedTime}")
    })
}
```

---

## SmbLicense

Signed license for CTS products.

```
data class SmbLicense(
    val product: String,
    val licensee: String,
    val expires: String,
    val signature: String
)
```

### Factory Methods

| Method | Description |
|---|---|
| `SmbLicense.fromFile(file: File)` | Parse from a `.lic` file on disk |
| `SmbLicense.fromStream(stream: InputStream)` | Parse from an input stream (resources, assets) |
| `SmbLicense.fromString(text: String)` | Parse from a raw text string |

See the Licensing section for full usage details.

---

## Exception Reference

All exceptions extend `SmbException` (which extends `RuntimeException`), allowing both broad and specific catch clauses.

### Exception Hierarchy

```
SmbException (abstract base)
├── SmbConnectionException
│   └── SmbConnectionLostException
├── SmbAuthenticationException
├── SmbAccessDeniedException
├── SmbFileNotFoundException
├── SmbShareNotFoundException
├── SmbFileExistsException
├── SmbDirectoryNotEmptyException
├── SmbDiskFullException
├── SmbProtocolException
└── SmbLicenseException
```

## Exception Details

| Exception | Description |
|---|---|
| `SmbConnectionException` | Server unreachable, connection refused, or DNS resolution failed |
| `SmbConnectionLostException` | Connection dropped during an active operation |
| `SmbAuthenticationException` | Server rejected the provided credentials |
| `SmbAccessDeniedException` | Insufficient permissions for the requested operation |
| `SmbFileNotFoundException` | File or directory does not exist at the specified path |
| `SmbShareNotFoundException` | The share name does not exist on the server |
| `SmbFileExistsException` | Target path already exists (e.g., during rename) |
| `SmbDirectoryNotEmptyException` | Cannot delete a non-empty directory with `delete()` |
| `SmbDiskFullException` | Server disk is full during a write operation |
| `SmbProtocolException` | Unexpected SMB protocol error with unmapped NT status code. The `statusCode: UInt` property contains the raw NT status code. |
| `SmbLicenseException` | License missing, invalid, expired, or product mismatch |

## Error Handling Example

```
try {
    SmbClient.connect("server", credentials, config = config).use { client
        ->
        client.connectShare("share").use { share ->
            val content = share.readFile("data.txt") {
        it.buffer().readUtf8() }
            println(content)
        }
    }
} catch (e: SmbLicenseException) {
    println("License error: ${e.message}")
} catch (e: SmbAuthenticationException) {
    println("Login failed: ${e.message}")
} catch (e: SmbConnectionException) {
    println("Cannot reach server: ${e.message}")
} catch (e: SmbFileNotFoundException) {
    println("File not found: ${e.message}")
} catch (e: SmbAccessDeniedException) {
    println("Permission denied: ${e.message}")
} catch (e: SmbException) {
    println("SMB error: ${e.message}")
}
```

---

## Complete Examples

### JVM: File Backup Script

```
import com.ctreesoft.smb.api.*
import kotlinx.coroutines.flow.toList
import kotlinx.coroutines.runBlocking
import okio.buffer
import okio.sink
import okio.source
import java.io.File

fun main() {
    val license = SmbLicense.fromStream(
        object {}::class.java.getResourceAsStream("/smb.lic")!!
    )
    val credentials = NtlmCredentials("backupuser", "s3cret", domain =
        "CORP")
    val config = SmbConfig(
        transferChunkSize = TransferChunkSize.ServerMax,
        license = license
    )

    runBlocking {
        SmbClient.connect("nas.corp.local", credentials, config =
        config).use { client ->
            client.connectShare("backups").use { share ->
                // Create today's backup directory
                val today = java.time.LocalDate.now().toString()
                share.createDirectory("daily/$today")

                // Upload local files
                val localDir = File("data")
                localDir.listFiles()?.forEach { file ->
                    println("Uploading ${file.name}...")
                    share.writeFile("daily/$today/${file.name}") { sink ->
                        val bufferedSink = sink.buffer()
                        file.source().buffer().use { src ->
                            bufferedSink.writeAll(src)
                        }
                        bufferedSink.flush()
                    }
                }

                // List what we uploaded
                val entries = share.listDirectory("daily/$today").toList()
                println("Backup complete: ${entries.size} files")
                entries.forEach { println("  ${it.name} (${it.size}
        bytes)") }

                // Clean up old backups
                val allDays = share.listDirectory("daily")
```

```
                    .filter { it.isDirectory }
                    .toList()
                    .sortedBy { it.name }

                if (allDays.size > 7) {
                    val toDelete = allDays.dropLast(7)
                    toDelete.forEach { dir ->
                        println("Deleting old backup: ${dir.name}")
                        share.deleteRecursively("daily/${dir.name}")
                    }
                }
            }
        }
    }
}
```

## Android: File Browser Activity

```
import android.os.Bundle
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.*
import com.ctreesoft.smb.api.*
import com.ctreesoft.smb.error.*
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.toList
import kotlinx.coroutines.launch
import okio.buffer
import okio.sink
import java.io.File

class SmbBrowserViewModel(private val app: Application) :
        AndroidViewModel(app) {

    private val _files = MutableStateFlow<List<SmbFileEntry>>(emptyList())
    val files: StateFlow<List<SmbFileEntry>> = _files

    private val _error = MutableStateFlow<String?>(null)
    val error: StateFlow<String?> = _error

    private var client: SmbClient? = null
    private var share: SmbShare? = null

    fun connect(host: String, user: String, pass: String, shareName:
        String) {
        viewModelScope.launch(Dispatchers.IO) {
            try {
                val license =
        SmbLicense.fromStream(app.assets.open("smb.lic"))
                val config = SmbConfig(license = license)
                val credentials = NtlmCredentials(user, pass)

                client = SmbClient.connect(host, credentials, config =
        config)
                share = client!!.connectShare(shareName)

                browse("/")
            } catch (e: SmbAuthenticationException) {
                _error.value = "Invalid username or password"
            } catch (e: SmbConnectionException) {
                _error.value = "Cannot reach server: ${e.message}"
            } catch (e: SmbLicenseException) {
                _error.value = "License error: ${e.message}"
            }
```

```
        }
    }

    fun browse(path: String) {
        viewModelScope.launch(Dispatchers.IO) {
            try {
                val entries = share!!.listDirectory(path).toList()
                _files.value = entries.sortedWith(
                    compareByDescending<SmbFileEntry> { it.isDirectory }
                        .thenBy { it.name.lowercase() }
                )
            } catch (e: SmbFileNotFoundException) {
                _error.value = "Directory not found: $path"
            } catch (e: SmbAccessDeniedException) {
                _error.value = "Access denied: $path"
            }
        }
    }

    fun downloadFile(remotePath: String) {
        viewModelScope.launch(Dispatchers.IO) {
            try {
                val fileName = remotePath.substringAfterLast("/")
                val localFile = File(app.filesDir, fileName)

                share!!.readFile(remotePath) { source ->
                    localFile.sink().buffer().use { localSink ->
                        localSink.writeAll(source)
                    }
                }
            } catch (e: SmbException) {
                _error.value = "Download failed: ${e.message}"
            }
        }
    }

    override fun onCleared() {
        share?.close()
        client?.close()
    }
}
```

## BlackBerry Dynamics Integration

```
import com.ctreesoft.smb.api.*
import com.good.gd.net.GDSocket

suspend fun connectViaBBD(host: String, user: String, pass: String):
        SmbClient {
    val license = SmbLicense.fromStream(
        javaClass.getResourceAsStream("/smb.lic")!!
    )

    val config = SmbConfig(
        socketFactory = SmbSocketFactory { GDSocket() },
        license = license
    )

    return SmbClient.connect(host, NtlmCredentials(user, pass), config =
        config)
}
```

### Server-Side: Processing Uploaded Files

```
import com.ctreesoft.smb.api.*
import kotlinx.coroutines.flow.filter
import kotlinx.coroutines.flow.toList
import okio.buffer

suspend fun processNewUploads(share: SmbShare) {
    // Find unprocessed CSV files
    val csvFiles = share.listDirectory("inbox")
        .filter { !it.isDirectory && it.name.endsWith(".csv") }
        .toList()

    for (file in csvFiles) {
        println("Processing ${file.name} (${file.size} bytes)...")

        // Read and process
        val content = share.readFile("inbox/${file.name}") { source ->
            source.buffer().readUtf8()
        }

        val lineCount = content.lines().size
        println("  $lineCount lines processed")

        // Move to processed folder
        share.createDirectory("processed")
        share.rename("inbox/${file.name}", "processed/${file.name}")
    }

    println("Done. Processed ${csvFiles.size} files.")
}
```

---

## Logging

The library uses SLF4J 1.7.x for logging. All log messages use the `com.ctreesoft.smb` logger namespace, prefixed with `SMB:` for easy filtering. No log output is produced unless you add an SLF4J backend.

### JVM

```
// For simple console logging
implementation("org.slf4j:slf4j-simple:1.7.36")

// Or for production logging
implementation("ch.qos.logback:logback-classic:1.4.14")
```

### Android

```
// Bridge SLF4J to Timber (no additional configuration needed)
implementation("com.arcao:slf4j-timber:3.1")
```

### Log Levels

| Level | Output |
|---|---|
| `INFO` | Connection lifecycle events (connect, disconnect, share open/close) |
| `DEBUG` | File operation progress, credit management, request/response flow |
| `TRACE` | Cryptographic key material and signature hex dumps. **Never use in production.** |

---

## Thread Safety

`SmbClient`, `SmbShare`, and all their operations are thread-safe. Multiple coroutines can call `connectShare()`, `listDirectory()`, `readFile()`, `writeFile()`, and all other operations concurrently on the same instance. The underlying session multiplexes requests over a single TCP connection using SMB2 message IDs and credit management.

---

# Platform Notes

## Minimum Requirements

| Platform | Minimum Version |
|---|---|
| JVM | Java 17+ |
| Android | API 26+ (Android 8.0) |
| Kotlin | 2.0+ |

## Android ProGuard / R8

The library's public API classes are not obfuscated and require no additional ProGuard rules. If you encounter issues, add:

```
-keep class com.ctreesoft.smb.api.** { *; }
-keep class com.ctreesoft.smb.error.** { *; }
```

---

Copyright 2026 Coconut Tree Software, Inc. All rights reserved.