| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/JavaCompiler.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/tools/ForwardingJavaFileObject.html)   [**NEXT CLASS**](http://docs.google.com/javax/tools/JavaCompiler.CompilationTask.html) | [**FRAMES**](http://docs.google.com/index.html?javax/tools/JavaCompiler.html)    [**NO FRAMES**](http://docs.google.com/JavaCompiler.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#3znysh7) | FIELD | CONSTR | [METHOD](#2et92p0) | DETAIL: FIELD | CONSTR | [METHOD](#1t3h5sf) |

## **javax.tools**

Interface JavaCompiler

**All Superinterfaces:** [OptionChecker](http://docs.google.com/javax/tools/OptionChecker.html), [Tool](http://docs.google.com/javax/tools/Tool.html)

public interface **JavaCompiler**extends [Tool](http://docs.google.com/javax/tools/Tool.html), [OptionChecker](http://docs.google.com/javax/tools/OptionChecker.html)

Interface to invoke Java™ programming language compilers from programs.

The compiler might generate diagnostics during compilation (for example, error messages). If a diagnostic listener is provided, the diagnostics will be supplied to the listener. If no listener is provided, the diagnostics will be formatted in an unspecified format and written to the default output, which is System.err unless otherwise specified. Even if a diagnostic listener is supplied, some diagnostics might not fit in a Diagnostic and will be written to the default output.

A compiler tool has an associated standard file manager, which is the file manager that is native to the tool (or built-in). The standard file manager can be obtained by calling [getStandardFileManager](http://docs.google.com/javax/tools/JavaCompiler.html#getStandardFileManager(javax.tools.DiagnosticListener,%20java.util.Locale,%20java.nio.charset.Charset)).

A compiler tool must function with any file manager as long as any additional requirements as detailed in the methods below are met. If no file manager is provided, the compiler tool will use a standard file manager such as the one returned by [getStandardFileManager](http://docs.google.com/javax/tools/JavaCompiler.html#getStandardFileManager(javax.tools.DiagnosticListener,%20java.util.Locale,%20java.nio.charset.Charset)).

An instance implementing this interface must conform to the Java Language Specification and generate class files conforming to the Java Virtual Machine specification. The versions of these specifications are defined in the [Tool](http://docs.google.com/javax/tools/Tool.html) interface. Additionally, an instance of this interface supporting [SourceVersion.RELEASE\_6](http://docs.google.com/javax/lang/model/SourceVersion.html#RELEASE_6) or higher must also support [annotation processing](http://docs.google.com/javax/annotation/processing/package-summary.html).

The compiler relies on two services: [diagnostic listener](http://docs.google.com/javax/tools/DiagnosticListener.html) and [file manager](http://docs.google.com/javax/tools/JavaFileManager.html). Although most classes and interfaces in this package defines an API for compilers (and tools in general) the interfaces [DiagnosticListener](http://docs.google.com/javax/tools/DiagnosticListener.html), [JavaFileManager](http://docs.google.com/javax/tools/JavaFileManager.html), [FileObject](http://docs.google.com/javax/tools/FileObject.html), and [JavaFileObject](http://docs.google.com/javax/tools/JavaFileObject.html) are not intended to be used in applications. Instead these interfaces are intended to be implemented and used to provide customized services for a compiler and thus defines an SPI for compilers.

There are a number of classes and interfaces in this package which are designed to ease the implementation of the SPI to customize the behavior of a compiler:

[StandardJavaFileManager](http://docs.google.com/javax/tools/StandardJavaFileManager.html) Every compiler which implements this interface provides a standard file manager for operating on regular [files](http://docs.google.com/java/io/File.html). The StandardJavaFileManager interface defines additional methods for creating file objects from regular files.

The standard file manager serves two purposes:

* basic building block for customizing how a compiler reads and writes files
* sharing between multiple compilation tasks

Reusing a file manager can potentially reduce overhead of scanning the file system and reading jar files. Although there might be no reduction in overhead, a standard file manager must work with multiple sequential compilations making the following example a recommended coding pattern:

Files[] files1 = ... ; // input for first compilation task  
 Files[] files2 = ... ; // input for second compilation task  
  
 JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();  
 StandardJavaFileManager fileManager = compiler.getStandardFileManager(null, null, null);  
  
 Iterable<? extends JavaFileObject> compilationUnits1 =  
 fileManager.getJavaFileObjectsFromFiles([Arrays.asList](http://docs.google.com/java/util/Arrays.html#asList(T...))(files1));  
 compiler.getTask(null, fileManager, null, null, null, compilationUnits1).call();  
  
 Iterable<? extends JavaFileObject> compilationUnits2 =  
 fileManager.getJavaFileObjects(files2); // use alternative method  
 // reuse the same file manager to allow caching of jar files  
 compiler.getTask(null, fileManager, null, null, null, compilationUnits2).call();  
  
 fileManager.close();

[DiagnosticCollector](http://docs.google.com/javax/tools/DiagnosticCollector.html) Used to collect diagnostics in a list, for example:

Iterable<? extends JavaFileObject> compilationUnits = ...;  
 JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();  
 DiagnosticCollector<JavaFileObject> diagnostics = new DiagnosticCollector<JavaFileObject>();  
 StandardJavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, null, null);  
 compiler.getTask(null, fileManager, diagnostics, null, null, compilationUnits).call();  
  
 for (Diagnostic diagnostic : diagnostics.getDiagnostics())  
 System.out.format("Error on line %d in %d%n",  
 diagnostic.getLineNumber()  
 diagnostic.getSource().toUri());  
  
 fileManager.close();

[ForwardingJavaFileManager](http://docs.google.com/javax/tools/ForwardingJavaFileManager.html), [ForwardingFileObject](http://docs.google.com/javax/tools/ForwardingFileObject.html), and [ForwardingJavaFileObject](http://docs.google.com/javax/tools/ForwardingJavaFileObject.html) Subclassing is not available for overriding the behavior of a standard file manager as it is created by calling a method on a compiler, not by invoking a constructor. Instead forwarding (or delegation) should be used. These classes makes it easy to forward most calls to a given file manager or file object while allowing customizing behavior. For example, consider how to log all calls to [JavaFileManager.flush()](http://docs.google.com/javax/tools/JavaFileManager.html#flush()):

final [Logger](http://docs.google.com/java/util/logging/Logger.html) logger = ...;  
 Iterable<? extends JavaFileObject> compilationUnits = ...;  
 JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();  
 StandardJavaFileManager stdFileManager = compiler.getStandardFileManager(null, null, null);  
 JavaFileManager fileManager = new ForwardingJavaFileManager(stdFileManager) {  
 public void flush() {  
 logger.entering(StandardJavaFileManager.class.getName(), "flush");  
 super.flush();  
 logger.exiting(StandardJavaFileManager.class.getName(), "flush");  
 }  
 };  
 compiler.getTask(null, fileManager, null, null, null, compilationUnits).call();

[SimpleJavaFileObject](http://docs.google.com/javax/tools/SimpleJavaFileObject.html) This class provides a basic file object implementation which can be used as building block for creating file objects. For example, here is how to define a file object which represent source code stored in a string:

/\*\*  
 \* A file object used to represent source coming from a string.  
 \*/  
 public class JavaSourceFromString extends SimpleJavaFileObject {  
 /\*\*  
 \* The source code of this "file".  
 \*/  
 final String code;  
  
 /\*\*  
 \* Constructs a new JavaSourceFromString.  
 \* @param name the name of the compilation unit represented by this file object  
 \* @param code the source code for the compilation unit represented by this file object  
 \*/  
 JavaSourceFromString(String name, String code) {  
 super([URI.create](http://docs.google.com/java/net/URI.html#create(java.lang.String))("string:///" + name.replace('.','/') + Kind.SOURCE.extension),  
 Kind.SOURCE);  
 this.code = code;  
 }  
  
 @Override  
 public CharSequence getCharContent(boolean ignoreEncodingErrors) {  
 return code;  
 }  
 }

**Since:** 1.6 **See Also:**[DiagnosticListener](http://docs.google.com/javax/tools/DiagnosticListener.html), [Diagnostic](http://docs.google.com/javax/tools/Diagnostic.html), [JavaFileManager](http://docs.google.com/javax/tools/JavaFileManager.html)

| **Nested Class Summary** | |
| --- | --- |
| static interface | [**JavaCompiler.CompilationTask**](http://docs.google.com/javax/tools/JavaCompiler.CompilationTask.html)            Interface representing a future for a compilation task. |

| **Method Summary** | |
| --- | --- |
| [StandardJavaFileManager](http://docs.google.com/javax/tools/StandardJavaFileManager.html) | [**getStandardFileManager**](http://docs.google.com/javax/tools/JavaCompiler.html#getStandardFileManager(javax.tools.DiagnosticListener,%20java.util.Locale,%20java.nio.charset.Charset))([DiagnosticListener](http://docs.google.com/javax/tools/DiagnosticListener.html)<? super [JavaFileObject](http://docs.google.com/javax/tools/JavaFileObject.html)> diagnosticListener, [Locale](http://docs.google.com/java/util/Locale.html) locale, [Charset](http://docs.google.com/java/nio/charset/Charset.html) charset)            Gets a new instance of the standard file manager implementation for this tool. |
| [JavaCompiler.CompilationTask](http://docs.google.com/javax/tools/JavaCompiler.CompilationTask.html) | [**getTask**](http://docs.google.com/javax/tools/JavaCompiler.html#getTask(java.io.Writer,%20javax.tools.JavaFileManager,%20javax.tools.DiagnosticListener,%20java.lang.Iterable,%20java.lang.Iterable,%20java.lang.Iterable))([Writer](http://docs.google.com/java/io/Writer.html) out, [JavaFileManager](http://docs.google.com/javax/tools/JavaFileManager.html) fileManager, [DiagnosticListener](http://docs.google.com/javax/tools/DiagnosticListener.html)<? super [JavaFileObject](http://docs.google.com/javax/tools/JavaFileObject.html)> diagnosticListener, [Iterable](http://docs.google.com/java/lang/Iterable.html)<[String](http://docs.google.com/java/lang/String.html)> options, [Iterable](http://docs.google.com/java/lang/Iterable.html)<[String](http://docs.google.com/java/lang/String.html)> classes, [Iterable](http://docs.google.com/java/lang/Iterable.html)<? extends [JavaFileObject](http://docs.google.com/javax/tools/JavaFileObject.html)> compilationUnits)            Creates a future for a compilation task with the given components and arguments. |

| **Methods inherited from interface javax.tools.**[**Tool**](http://docs.google.com/javax/tools/Tool.html) |
| --- |
| [getSourceVersions](http://docs.google.com/javax/tools/Tool.html#getSourceVersions()), [run](http://docs.google.com/javax/tools/Tool.html#run(java.io.InputStream,%20java.io.OutputStream,%20java.io.OutputStream,%20java.lang.String...)) |

| **Methods inherited from interface javax.tools.**[**OptionChecker**](http://docs.google.com/javax/tools/OptionChecker.html) |
| --- |
| [isSupportedOption](http://docs.google.com/javax/tools/OptionChecker.html#isSupportedOption(java.lang.String)) |

| **Method Detail** |
| --- |

### getTask

[JavaCompiler.CompilationTask](http://docs.google.com/javax/tools/JavaCompiler.CompilationTask.html) **getTask**([Writer](http://docs.google.com/java/io/Writer.html) out,  
 [JavaFileManager](http://docs.google.com/javax/tools/JavaFileManager.html) fileManager,  
 [DiagnosticListener](http://docs.google.com/javax/tools/DiagnosticListener.html)<? super [JavaFileObject](http://docs.google.com/javax/tools/JavaFileObject.html)> diagnosticListener,  
 [Iterable](http://docs.google.com/java/lang/Iterable.html)<[String](http://docs.google.com/java/lang/String.html)> options,  
 [Iterable](http://docs.google.com/java/lang/Iterable.html)<[String](http://docs.google.com/java/lang/String.html)> classes,  
 [Iterable](http://docs.google.com/java/lang/Iterable.html)<? extends [JavaFileObject](http://docs.google.com/javax/tools/JavaFileObject.html)> compilationUnits)

Creates a future for a compilation task with the given components and arguments. The compilation might not have completed as described in the CompilationTask interface.

If a file manager is provided, it must be able to handle all locations defined in [StandardLocation](http://docs.google.com/javax/tools/StandardLocation.html).

**Parameters:**out - a Writer for additional output from the compiler; use System.err if nullfileManager - a file manager; if null use the compiler's standard filemanagerdiagnosticListener - a diagnostic listener; if null use the compiler's default method for reporting diagnosticsoptions - compiler options, null means no optionsclasses - class names (for annotation processing), null means no class namescompilationUnits - the compilation units to compile, null means no compilation units **Returns:**an object representing the compilation **Throws:** [RuntimeException](http://docs.google.com/java/lang/RuntimeException.html) - if an unrecoverable error occurred in a user supplied component. The [cause](http://docs.google.com/java/lang/Throwable.html#getCause()) will be the error in user code. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if any of the given compilation units are of other kind than [source](http://docs.google.com/javax/tools/JavaFileObject.Kind.html#SOURCE)

### getStandardFileManager

[StandardJavaFileManager](http://docs.google.com/javax/tools/StandardJavaFileManager.html) **getStandardFileManager**([DiagnosticListener](http://docs.google.com/javax/tools/DiagnosticListener.html)<? super [JavaFileObject](http://docs.google.com/javax/tools/JavaFileObject.html)> diagnosticListener,  
 [Locale](http://docs.google.com/java/util/Locale.html) locale,  
 [Charset](http://docs.google.com/java/nio/charset/Charset.html) charset)

Gets a new instance of the standard file manager implementation for this tool. The file manager will use the given diagnostic listener for producing any non-fatal diagnostics. Fatal errors will be signalled with the appropriate exceptions.

The standard file manager will be automatically reopened if it is accessed after calls to flush or close. The standard file manager must be usable with other tools.

**Parameters:**diagnosticListener - a diagnostic listener for non-fatal diagnostics; if null use the compiler's default method for reporting diagnosticslocale - the locale to apply when formatting diagnostics; null means the [default locale](http://docs.google.com/java/util/Locale.html#getDefault()).charset - the character set used for decoding bytes; if null use the platform default **Returns:**the standard file manager

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/JavaCompiler.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/tools/ForwardingJavaFileObject.html)   [**NEXT CLASS**](http://docs.google.com/javax/tools/JavaCompiler.CompilationTask.html) | [**FRAMES**](http://docs.google.com/index.html?javax/tools/JavaCompiler.html)    [**NO FRAMES**](http://docs.google.com/JavaCompiler.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: [NESTED](#3znysh7) | FIELD | CONSTR | [METHOD](#2et92p0) | DETAIL: FIELD | CONSTR | [METHOD](#1t3h5sf) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).