Documentation

The Java™ Tutorials

Trail: Essential Java Classes
Lesson: The Platform Environment
Section: System Utilities

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

System Properties

In Properties, we examined the way an application can use `Properties` objects to maintain its configuration. The Java platform itself uses a `Properties` object to maintain its own configuration. The `System` class maintains a `Properties` object that describes the configuration of the current working environment. System properties include information about the current user, the current version of the Java runtime, and the character used to separate components of a file path name.

The following table describes some of the most important system properties

Key	Meaning
`"file.separator"`	Character that separates components of a file path. This is "`/`" on UNIX and "`\`" on Windows.
`"java.class.path"`	Path used to find directories and JAR archives containing class files. Elements of the class path are separated by a platform-specific character specified in the `path.separator` property.
`"java.home"`	Installation directory for Java Runtime Environment (JRE)
`"java.vendor"`	JRE vendor name
`"java.vendor.url"`	JRE vendor URL
`"java.version"`	JRE version number
`"line.separator"`	Sequence used by operating system to separate lines in text files
`"os.arch"`	Operating system architecture
`"os.name"`	Operating system name
`"os.version"`	Operating system version
`"path.separator"`	Path separator character used in `java.class.path`
`"user.dir"`	User working directory
`"user.home"`	User home directory
`"user.name"`	User account name

Security consideration: Access to system properties can be restricted by the Security Manager. This is most often an issue in applets, which are prevented from reading some system properties, and from writing *any* system properties. For more on accessing system properties in applets, refer to System Properties in the Doing More With Java Rich Internet Applications lesson.

Reading System Properties

The `System` class has two methods used to read system properties: `getProperty` and `getProperties`.

The `System` class has two different versions of `getProperty`. Both retrieve the value of the property named in the argument list. The simpler of the two `getProperty` methods takes a single argument, a property key For example, to get the value of `path.separator`, use the following statement:

```
System.getProperty("path.separator");
```

The `getProperty` method returns a string containing the value of the property. If the property does not exist, this version of `getProperty` returns null.

The other version of `getProperty` requires two `String` arguments: the first argument is the key to look up and the second argument is a default value to return if the key cannot be found or if it has no value. For example, the following invocation of `getProperty` looks up the `System` property

called `subliminal.message`. This is not a valid system property, so instead of returning null, this method returns the default value provided as a second argument: `"Buy StayPuft Marshmallows!"`

```
System.getProperty("subliminal.message", "Buy StayPuft Marshmallows!");
```

The last method provided by the `System` class to access property values is the `getProperties` method, which returns a `Properties` object. This object contains a complete set of system property definitions.

Writing System Properties

To modify the existing set of system properties, use `System.setProperties`. This method takes a `Properties` object that has been initialized to contain the properties to be set. This method replaces the entire set of system properties with the new set represented by the `Properties` object.

Warning: Changing system properties is potentially dangerous and should be done with discretion. Many system properties are not reread after start-up and are there for informational purposes. Changing some properties may have unexpected side-effects.

The next example, `PropertiesTest`, creates a `Properties` object and initializes it from `myProperties.txt` .

```
subliminal.message=Buy StayPuft Marshmallows!
```

`PropertiesTest` then uses `System.setProperties` to install the new `Properties` objects as the current set of system properties.

```
import java.io.FileInputStream;
import java.util.Properties;

public class PropertiesTest {
    public static void main(String[] args)
        throws Exception {

        // set up new properties object
        // from file "myProperties.txt"
        FileInputStream propFile =
            new FileInputStream( "myProperties.txt");
        Properties p =
            new Properties(System.getProperties());
        p.load(propFile);

        // set the system properties
        System.setProperties(p);
        // display new properties
        System.getProperties().list(System.out);
    }
}
```

Note how `PropertiesTest` creates the `Properties` object, `p`, which is used as the argument to `setProperties`:

```
Properties p = new Properties(System.getProperties());
```

This statement initializes the new properties object, `p`, with the current set of system properties, which in the case of this small application, is the set of properties initialized by the runtime system. Then the application loads additional properties into `p` from the file `myProperties.txt` and sets the system properties to `p`. This has the effect of adding the properties listed in `myProperties.txt` to the set of properties created by the runtime system at startup. Note that an application can create `p` without any default `Properties` object, like this:

```
Properties p = new Properties();
```

Also note that the value of system properties can be overwritten! For example, if `myProperties.txt` contains the following line, the `java.vendor` system property will be overwritten:

```
java.vendor=Acme Software Company
```

In general, be careful not to overwrite system properties.

The `setProperties` method changes the set of system properties for the current running application. These changes are not persistent. That is, changing the system properties within an application will not affect future invocations of the Java interpreter for this or any other application. The runtime system re-initializes the system properties each time its starts up. If changes to system properties are to be persistent, then the application must write the values to some file before exiting and read them in again upon startup.

Previous page: Command-Line I/O Objects
Next page: The Security Manager