| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Preferences.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/java/util/prefs/PreferenceChangeListener.html)   [**NEXT CLASS**](http://docs.google.com/java/util/prefs/PreferencesFactory.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/prefs/Preferences.html)    [**NO FRAMES**](http://docs.google.com/Preferences.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#3rdcrjn) | [METHOD](#lnxbz9) |

## **java.util.prefs**

Class Preferences

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **java.util.prefs.Preferences**

**Direct Known Subclasses:** [AbstractPreferences](http://docs.google.com/java/util/prefs/AbstractPreferences.html)

public abstract class **Preferences**extends [Object](http://docs.google.com/java/lang/Object.html)

A node in a hierarchical collection of preference data. This class allows applications to store and retrieve user and system preference and configuration data. This data is stored persistently in an implementation-dependent backing store. Typical implementations include flat files, OS-specific registries, directory servers and SQL databases. The user of this class needn't be concerned with details of the backing store.

There are two separate trees of preference nodes, one for user preferences and one for system preferences. Each user has a separate user preference tree, and all users in a given system share the same system preference tree. The precise description of "user" and "system" will vary from implementation to implementation. Typical information stored in the user preference tree might include font choice, color choice, or preferred window location and size for a particular application. Typical information stored in the system preference tree might include installation configuration data for an application.

Nodes in a preference tree are named in a similar fashion to directories in a hierarchical file system. Every node in a preference tree has a *node name* (which is not necessarily unique), a unique *absolute path name*, and a path name *relative* to each ancestor including itself.

The root node has a node name of the empty string (""). Every other node has an arbitrary node name, specified at the time it is created. The only restrictions on this name are that it cannot be the empty string, and it cannot contain the slash character ('/').

The root node has an absolute path name of "/". Children of the root node have absolute path names of "/" + *<node name>*. All other nodes have absolute path names of *<parent's absolute path name>* + "/" + *<node name>*. Note that all absolute path names begin with the slash character.

A node *n*'s path name relative to its ancestor *a* is simply the string that must be appended to *a*'s absolute path name in order to form *n*'s absolute path name, with the initial slash character (if present) removed. Note that:

* No relative path names begin with the slash character.
* Every node's path name relative to itself is the empty string.
* Every node's path name relative to its parent is its node name (except for the root node, which does not have a parent).
* Every node's path name relative to the root is its absolute path name with the initial slash character removed.

Note finally that:

* No path name contains multiple consecutive slash characters.
* No path name with the exception of the root's absolute path name ends in the slash character.
* Any string that conforms to these two rules is a valid path name.

All of the methods that modify preferences data are permitted to operate asynchronously; they may return immediately, and changes will eventually propagate to the persistent backing store with an implementation-dependent delay. The flush method may be used to synchronously force updates to the backing store. Normal termination of the Java Virtual Machine will *not* result in the loss of pending updates -- an explicit flush invocation is *not* required upon termination to ensure that pending updates are made persistent.

All of the methods that read preferences from a Preferences object require the invoker to provide a default value. The default value is returned if no value has been previously set *or if the backing store is unavailable*. The intent is to allow applications to operate, albeit with slightly degraded functionality, even if the backing store becomes unavailable. Several methods, like flush, have semantics that prevent them from operating if the backing store is unavailable. Ordinary applications should have no need to invoke any of these methods, which can be identified by the fact that they are declared to throw [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html).

The methods in this class may be invoked concurrently by multiple threads in a single JVM without the need for external synchronization, and the results will be equivalent to some serial execution. If this class is used concurrently *by multiple JVMs* that store their preference data in the same backing store, the data store will not be corrupted, but no other guarantees are made concerning the consistency of the preference data.

This class contains an export/import facility, allowing preferences to be "exported" to an XML document, and XML documents representing preferences to be "imported" back into the system. This facility may be used to back up all or part of a preference tree, and subsequently restore from the backup.

The XML document has the following DOCTYPE declaration:

<!DOCTYPE preferences SYSTEM "http://java.sun.com/dtd/preferences.dtd">

Note that the system URI (http://java.sun.com/dtd/preferences.dtd) is *not* accessed when exporting or importing preferences; it merely serves as a string to uniquely identify the DTD, which is:

<?xml version="1.0" encoding="UTF-8"?>  
  
 <!-- DTD for a Preferences tree. -->  
  
 <!-- The preferences element is at the root of an XML document  
 representing a Preferences tree. -->  
 <!ELEMENT preferences (root)>  
   
 <!-- The preferences element contains an optional version attribute,  
 which specifies version of DTD. -->  
 <!ATTLIST preferences EXTERNAL\_XML\_VERSION CDATA "0.0" >   
  
 <!-- The root element has a map representing the root's preferences  
 (if any), and one node for each child of the root (if any). -->  
 <!ELEMENT root (map, node\*) >  
  
 <!-- Additionally, the root contains a type attribute, which  
 specifies whether it's the system or user root. -->  
 <!ATTLIST root  
 type (system|user) #REQUIRED >  
  
 <!-- Each node has a map representing its preferences (if any),  
 and one node for each child (if any). -->  
 <!ELEMENT node (map, node\*) >  
  
 <!-- Additionally, each node has a name attribute -->  
 <!ATTLIST node  
 name CDATA #REQUIRED >  
  
 <!-- A map represents the preferences stored at a node (if any). -->  
 <!ELEMENT map (entry\*) >  
  
 <!-- An entry represents a single preference, which is simply  
 a key-value pair. -->  
 <!ELEMENT entry EMPTY >  
 <!ATTLIST entry  
 key CDATA #REQUIRED  
 value CDATA #REQUIRED >

Every Preferences implementation must have an associated [PreferencesFactory](http://docs.google.com/java/util/prefs/PreferencesFactory.html) implementation. Every Java(TM) SE implementation must provide some means of specifying which PreferencesFactory implementation is used to generate the root preferences nodes. This allows the administrator to replace the default preferences implementation with an alternative implementation.

Implementation note: In Sun's JRE, the PreferencesFactory implementation is located as follows:

1. If the system property java.util.prefs.PreferencesFactory is defined, then it is taken to be the fully-qualified name of a class implementing the PreferencesFactory interface. The class is loaded and instantiated; if this process fails then an unspecified error is thrown.
2. If a PreferencesFactory implementation class file has been installed in a jar file that is visible to the [system class loader](http://docs.google.com/java/lang/ClassLoader.html#getSystemClassLoader()), and that jar file contains a provider-configuration file named java.util.prefs.PreferencesFactory in the resource directory META-INF/services, then the first class name specified in that file is taken. If more than one such jar file is provided, the first one found will be used. The class is loaded and instantiated; if this process fails then an unspecified error is thrown.
3. Finally, if neither the above-mentioned system property nor an extension jar file is provided, then the system-wide default PreferencesFactory implementation for the underlying platform is loaded and instantiated.

**Since:** 1.4

| **Field Summary** | |
| --- | --- |
| static int | [**MAX\_KEY\_LENGTH**](http://docs.google.com/java/util/prefs/Preferences.html#MAX_KEY_LENGTH)            Maximum length of string allowed as a key (80 characters). |
| static int | [**MAX\_NAME\_LENGTH**](http://docs.google.com/java/util/prefs/Preferences.html#MAX_NAME_LENGTH)            Maximum length of a node name (80 characters). |
| static int | [**MAX\_VALUE\_LENGTH**](http://docs.google.com/java/util/prefs/Preferences.html#MAX_VALUE_LENGTH)            Maximum length of string allowed as a value (8192 characters). |

| **Constructor Summary** | |
| --- | --- |
| protected | [**Preferences**](http://docs.google.com/java/util/prefs/Preferences.html#Preferences())()            Sole constructor. |

| **Method Summary** | |
| --- | --- |
| abstract  [String](http://docs.google.com/java/lang/String.html) | [**absolutePath**](http://docs.google.com/java/util/prefs/Preferences.html#absolutePath())()            Returns this preference node's absolute path name. |
| abstract  void | [**addNodeChangeListener**](http://docs.google.com/java/util/prefs/Preferences.html#addNodeChangeListener(java.util.prefs.NodeChangeListener))([NodeChangeListener](http://docs.google.com/java/util/prefs/NodeChangeListener.html) ncl)            Registers the specified listener to receive *node change events* for this node. |
| abstract  void | [**addPreferenceChangeListener**](http://docs.google.com/java/util/prefs/Preferences.html#addPreferenceChangeListener(java.util.prefs.PreferenceChangeListener))([PreferenceChangeListener](http://docs.google.com/java/util/prefs/PreferenceChangeListener.html) pcl)            Registers the specified listener to receive *preference change events* for this preference node. |
| abstract  [String](http://docs.google.com/java/lang/String.html)[] | [**childrenNames**](http://docs.google.com/java/util/prefs/Preferences.html#childrenNames())()            Returns the names of the children of this preference node, relative to this node. |
| abstract  void | [**clear**](http://docs.google.com/java/util/prefs/Preferences.html#clear())()            Removes all of the preferences (key-value associations) in this preference node. |
| abstract  void | [**exportNode**](http://docs.google.com/java/util/prefs/Preferences.html#exportNode(java.io.OutputStream))([OutputStream](http://docs.google.com/java/io/OutputStream.html) os)            Emits on the specified output stream an XML document representing all of the preferences contained in this node (but not its descendants). |
| abstract  void | [**exportSubtree**](http://docs.google.com/java/util/prefs/Preferences.html#exportSubtree(java.io.OutputStream))([OutputStream](http://docs.google.com/java/io/OutputStream.html) os)            Emits an XML document representing all of the preferences contained in this node and all of its descendants. |
| abstract  void | [**flush**](http://docs.google.com/java/util/prefs/Preferences.html#flush())()            Forces any changes in the contents of this preference node and its descendants to the persistent store. |
| abstract  [String](http://docs.google.com/java/lang/String.html) | [**get**](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String))([String](http://docs.google.com/java/lang/String.html) key, [String](http://docs.google.com/java/lang/String.html) def)            Returns the value associated with the specified key in this preference node. |
| abstract  boolean | [**getBoolean**](http://docs.google.com/java/util/prefs/Preferences.html#getBoolean(java.lang.String,%20boolean))([String](http://docs.google.com/java/lang/String.html) key, boolean def)            Returns the boolean value represented by the string associated with the specified key in this preference node. |
| abstract  byte[] | [**getByteArray**](http://docs.google.com/java/util/prefs/Preferences.html#getByteArray(java.lang.String,%20byte%5B%5D))([String](http://docs.google.com/java/lang/String.html) key, byte[] def)            Returns the byte array value represented by the string associated with the specified key in this preference node. |
| abstract  double | [**getDouble**](http://docs.google.com/java/util/prefs/Preferences.html#getDouble(java.lang.String,%20double))([String](http://docs.google.com/java/lang/String.html) key, double def)            Returns the double value represented by the string associated with the specified key in this preference node. |
| abstract  float | [**getFloat**](http://docs.google.com/java/util/prefs/Preferences.html#getFloat(java.lang.String,%20float))([String](http://docs.google.com/java/lang/String.html) key, float def)            Returns the float value represented by the string associated with the specified key in this preference node. |
| abstract  int | [**getInt**](http://docs.google.com/java/util/prefs/Preferences.html#getInt(java.lang.String,%20int))([String](http://docs.google.com/java/lang/String.html) key, int def)            Returns the int value represented by the string associated with the specified key in this preference node. |
| abstract  long | [**getLong**](http://docs.google.com/java/util/prefs/Preferences.html#getLong(java.lang.String,%20long))([String](http://docs.google.com/java/lang/String.html) key, long def)            Returns the long value represented by the string associated with the specified key in this preference node. |
| static void | [**importPreferences**](http://docs.google.com/java/util/prefs/Preferences.html#importPreferences(java.io.InputStream))([InputStream](http://docs.google.com/java/io/InputStream.html) is)            Imports all of the preferences represented by the XML document on the specified input stream. |
| abstract  boolean | [**isUserNode**](http://docs.google.com/java/util/prefs/Preferences.html#isUserNode())()            Returns true if this preference node is in the user preference tree, false if it's in the system preference tree. |
| abstract  [String](http://docs.google.com/java/lang/String.html)[] | [**keys**](http://docs.google.com/java/util/prefs/Preferences.html#keys())()            Returns all of the keys that have an associated value in this preference node. |
| abstract  [String](http://docs.google.com/java/lang/String.html) | [**name**](http://docs.google.com/java/util/prefs/Preferences.html#name())()            Returns this preference node's name, relative to its parent. |
| abstract  [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) | [**node**](http://docs.google.com/java/util/prefs/Preferences.html#node(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pathName)            Returns the named preference node in the same tree as this node, creating it and any of its ancestors if they do not already exist. |
| abstract  boolean | [**nodeExists**](http://docs.google.com/java/util/prefs/Preferences.html#nodeExists(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pathName)            Returns true if the named preference node exists in the same tree as this node. |
| abstract  [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) | [**parent**](http://docs.google.com/java/util/prefs/Preferences.html#parent())()            Returns the parent of this preference node, or null if this is the root. |
| abstract  void | [**put**](http://docs.google.com/java/util/prefs/Preferences.html#put(java.lang.String,%20java.lang.String))([String](http://docs.google.com/java/lang/String.html) key, [String](http://docs.google.com/java/lang/String.html) value)            Associates the specified value with the specified key in this preference node. |
| abstract  void | [**putBoolean**](http://docs.google.com/java/util/prefs/Preferences.html#putBoolean(java.lang.String,%20boolean))([String](http://docs.google.com/java/lang/String.html) key, boolean value)            Associates a string representing the specified boolean value with the specified key in this preference node. |
| abstract  void | [**putByteArray**](http://docs.google.com/java/util/prefs/Preferences.html#putByteArray(java.lang.String,%20byte%5B%5D))([String](http://docs.google.com/java/lang/String.html) key, byte[] value)            Associates a string representing the specified byte array with the specified key in this preference node. |
| abstract  void | [**putDouble**](http://docs.google.com/java/util/prefs/Preferences.html#putDouble(java.lang.String,%20double))([String](http://docs.google.com/java/lang/String.html) key, double value)            Associates a string representing the specified double value with the specified key in this preference node. |
| abstract  void | [**putFloat**](http://docs.google.com/java/util/prefs/Preferences.html#putFloat(java.lang.String,%20float))([String](http://docs.google.com/java/lang/String.html) key, float value)            Associates a string representing the specified float value with the specified key in this preference node. |
| abstract  void | [**putInt**](http://docs.google.com/java/util/prefs/Preferences.html#putInt(java.lang.String,%20int))([String](http://docs.google.com/java/lang/String.html) key, int value)            Associates a string representing the specified int value with the specified key in this preference node. |
| abstract  void | [**putLong**](http://docs.google.com/java/util/prefs/Preferences.html#putLong(java.lang.String,%20long))([String](http://docs.google.com/java/lang/String.html) key, long value)            Associates a string representing the specified long value with the specified key in this preference node. |
| abstract  void | [**remove**](http://docs.google.com/java/util/prefs/Preferences.html#remove(java.lang.String))([String](http://docs.google.com/java/lang/String.html) key)            Removes the value associated with the specified key in this preference node, if any. |
| abstract  void | [**removeNode**](http://docs.google.com/java/util/prefs/Preferences.html#removeNode())()            Removes this preference node and all of its descendants, invalidating any preferences contained in the removed nodes. |
| abstract  void | [**removeNodeChangeListener**](http://docs.google.com/java/util/prefs/Preferences.html#removeNodeChangeListener(java.util.prefs.NodeChangeListener))([NodeChangeListener](http://docs.google.com/java/util/prefs/NodeChangeListener.html) ncl)            Removes the specified NodeChangeListener, so it no longer receives change events. |
| abstract  void | [**removePreferenceChangeListener**](http://docs.google.com/java/util/prefs/Preferences.html#removePreferenceChangeListener(java.util.prefs.PreferenceChangeListener))([PreferenceChangeListener](http://docs.google.com/java/util/prefs/PreferenceChangeListener.html) pcl)            Removes the specified preference change listener, so it no longer receives preference change events. |
| abstract  void | [**sync**](http://docs.google.com/java/util/prefs/Preferences.html#sync())()            Ensures that future reads from this preference node and its descendants reflect any changes that were committed to the persistent store (from any VM) prior to the sync invocation. |
| static [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) | [**systemNodeForPackage**](http://docs.google.com/java/util/prefs/Preferences.html#systemNodeForPackage(java.lang.Class))([Class](http://docs.google.com/java/lang/Class.html)<?> c)            Returns the preference node from the system preference tree that is associated (by convention) with the specified class's package. |
| static [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) | [**systemRoot**](http://docs.google.com/java/util/prefs/Preferences.html#systemRoot())()            Returns the root preference node for the system. |
| abstract  [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/prefs/Preferences.html#toString())()            Returns a string representation of this preferences node, as if computed by the expression:(this.isUserNode() ? "User" : "System") + " Preference Node: " + this.absolutePath(). |
| static [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) | [**userNodeForPackage**](http://docs.google.com/java/util/prefs/Preferences.html#userNodeForPackage(java.lang.Class))([Class](http://docs.google.com/java/lang/Class.html)<?> c)            Returns the preference node from the calling user's preference tree that is associated (by convention) with the specified class's package. |
| static [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) | [**userRoot**](http://docs.google.com/java/util/prefs/Preferences.html#userRoot())()            Returns the root preference node for the calling user. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Field Detail** |
| --- |

### MAX\_KEY\_LENGTH

public static final int **MAX\_KEY\_LENGTH**

Maximum length of string allowed as a key (80 characters).

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.util.prefs.Preferences.MAX_KEY_LENGTH)

### MAX\_VALUE\_LENGTH

public static final int **MAX\_VALUE\_LENGTH**

Maximum length of string allowed as a value (8192 characters).

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.util.prefs.Preferences.MAX_VALUE_LENGTH)

### MAX\_NAME\_LENGTH

public static final int **MAX\_NAME\_LENGTH**

Maximum length of a node name (80 characters).

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.util.prefs.Preferences.MAX_NAME_LENGTH)

| **Constructor Detail** |
| --- |

### Preferences

protected **Preferences**()

Sole constructor. (For invocation by subclass constructors, typically implicit.)

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

### userNodeForPackage

public static [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) **userNodeForPackage**([Class](http://docs.google.com/java/lang/Class.html)<?> c)

Returns the preference node from the calling user's preference tree that is associated (by convention) with the specified class's package. The convention is as follows: the absolute path name of the node is the fully qualified package name, preceded by a slash ('/'), and with each period ('.') replaced by a slash. For example the absolute path name of the node associated with the class com.acme.widget.Foo is /com/acme/widget.

This convention does not apply to the unnamed package, whose associated preference node is <unnamed>. This node is not intended for long term use, but for convenience in the early development of programs that do not yet belong to a package, and for "throwaway" programs. *Valuable data should not be stored at this node as it is shared by all programs that use it.*

A class Foo wishing to access preferences pertaining to its package can obtain a preference node as follows:

static Preferences prefs = Preferences.userNodeForPackage(Foo.class);

This idiom obviates the need for using a string to describe the preferences node and decreases the likelihood of a run-time failure. (If the class name is misspelled, it will typically result in a compile-time error.)

Invoking this method will result in the creation of the returned node and its ancestors if they do not already exist. If the returned node did not exist prior to this call, this node and any ancestors that were created by this call are not guaranteed to become permanent until the flush method is called on the returned node (or one of its ancestors or descendants).

**Parameters:**c - the class for whose package a user preference node is desired. **Returns:**the user preference node associated with the package of which c is a member. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if c is null. [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if a security manager is present and it denies RuntimePermission("preferences").**See Also:**[RuntimePermission](http://docs.google.com/java/lang/RuntimePermission.html)

### systemNodeForPackage

public static [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) **systemNodeForPackage**([Class](http://docs.google.com/java/lang/Class.html)<?> c)

Returns the preference node from the system preference tree that is associated (by convention) with the specified class's package. The convention is as follows: the absolute path name of the node is the fully qualified package name, preceded by a slash ('/'), and with each period ('.') replaced by a slash. For example the absolute path name of the node associated with the class com.acme.widget.Foo is /com/acme/widget.

This convention does not apply to the unnamed package, whose associated preference node is <unnamed>. This node is not intended for long term use, but for convenience in the early development of programs that do not yet belong to a package, and for "throwaway" programs. *Valuable data should not be stored at this node as it is shared by all programs that use it.*

A class Foo wishing to access preferences pertaining to its package can obtain a preference node as follows:

static Preferences prefs = Preferences.systemNodeForPackage(Foo.class);

This idiom obviates the need for using a string to describe the preferences node and decreases the likelihood of a run-time failure. (If the class name is misspelled, it will typically result in a compile-time error.)

Invoking this method will result in the creation of the returned node and its ancestors if they do not already exist. If the returned node did not exist prior to this call, this node and any ancestors that were created by this call are not guaranteed to become permanent until the flush method is called on the returned node (or one of its ancestors or descendants).

**Parameters:**c - the class for whose package a system preference node is desired. **Returns:**the system preference node associated with the package of which c is a member. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if c is null. [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - if a security manager is present and it denies RuntimePermission("preferences").**See Also:**[RuntimePermission](http://docs.google.com/java/lang/RuntimePermission.html)

### userRoot

public static [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) **userRoot**()

Returns the root preference node for the calling user.

**Returns:**the root preference node for the calling user. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - If a security manager is present and it denies RuntimePermission("preferences").**See Also:**[RuntimePermission](http://docs.google.com/java/lang/RuntimePermission.html)

### systemRoot

public static [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) **systemRoot**()

Returns the root preference node for the system.

**Returns:**the root preference node for the system. **Throws:** [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - If a security manager is present and it denies RuntimePermission("preferences").**See Also:**[RuntimePermission](http://docs.google.com/java/lang/RuntimePermission.html)

### put

public abstract void **put**([String](http://docs.google.com/java/lang/String.html) key,  
 [String](http://docs.google.com/java/lang/String.html) value)

Associates the specified value with the specified key in this preference node.

**Parameters:**key - key with which the specified value is to be associated.value - value to be associated with the specified key. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key or value is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if key.length() exceeds MAX\_KEY\_LENGTH or if value.length exceeds MAX\_VALUE\_LENGTH. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.

### get

public abstract [String](http://docs.google.com/java/lang/String.html) **get**([String](http://docs.google.com/java/lang/String.html) key,  
 [String](http://docs.google.com/java/lang/String.html) def)

Returns the value associated with the specified key in this preference node. Returns the specified default if there is no value associated with the key, or the backing store is inaccessible.

Some implementations may store default values in their backing stores. If there is no value associated with the specified key but there is such a *stored default*, it is returned in preference to the specified default.

**Parameters:**key - key whose associated value is to be returned.def - the value to be returned in the event that this preference node has no value associated with key. **Returns:**the value associated with key, or def if no value is associated with key, or the backing store is inaccessible. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null. (A null value for def *is* permitted.)

### remove

public abstract void **remove**([String](http://docs.google.com/java/lang/String.html) key)

Removes the value associated with the specified key in this preference node, if any.

If this implementation supports *stored defaults*, and there is such a default for the specified preference, the stored default will be "exposed" by this call, in the sense that it will be returned by a succeeding call to get.

**Parameters:**key - key whose mapping is to be removed from the preference node. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.

### clear

public abstract void **clear**()  
 throws [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Removes all of the preferences (key-value associations) in this preference node. This call has no effect on any descendants of this node.

If this implementation supports *stored defaults*, and this node in the preferences hierarchy contains any such defaults, the stored defaults will be "exposed" by this call, in the sense that they will be returned by succeeding calls to get.

**Throws:** [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode())

### putInt

public abstract void **putInt**([String](http://docs.google.com/java/lang/String.html) key,  
 int value)

Associates a string representing the specified int value with the specified key in this preference node. The associated string is the one that would be returned if the int value were passed to [Integer.toString(int)](http://docs.google.com/java/lang/Integer.html#toString(int)). This method is intended for use in conjunction with [getInt(java.lang.String, int)](http://docs.google.com/java/util/prefs/Preferences.html#getInt(java.lang.String,%20int)).

**Parameters:**key - key with which the string form of value is to be associated.value - value whose string form is to be associated with key. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if key.length() exceeds MAX\_KEY\_LENGTH. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[getInt(String,int)](http://docs.google.com/java/util/prefs/Preferences.html#getInt(java.lang.String,%20int))

### getInt

public abstract int **getInt**([String](http://docs.google.com/java/lang/String.html) key,  
 int def)

Returns the int value represented by the string associated with the specified key in this preference node. The string is converted to an integer as by [Integer.parseInt(String)](http://docs.google.com/java/lang/Integer.html#parseInt(java.lang.String)). Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if Integer.parseInt(String) would throw a [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html) if the associated value were passed. This method is intended for use in conjunction with [putInt(java.lang.String, int)](http://docs.google.com/java/util/prefs/Preferences.html#putInt(java.lang.String,%20int)).

If the implementation supports *stored defaults* and such a default exists, is accessible, and could be converted to an int with Integer.parseInt, this int is returned in preference to the specified default.

**Parameters:**key - key whose associated value is to be returned as an int.def - the value to be returned in the event that this preference node has no value associated with key or the associated value cannot be interpreted as an int, or the backing store is inaccessible. **Returns:**the int value represented by the string associated with key in this preference node, or def if the associated value does not exist or cannot be interpreted as an int. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null.**See Also:**[putInt(String,int)](http://docs.google.com/java/util/prefs/Preferences.html#putInt(java.lang.String,%20int)), [get(String,String)](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String))

### putLong

public abstract void **putLong**([String](http://docs.google.com/java/lang/String.html) key,  
 long value)

Associates a string representing the specified long value with the specified key in this preference node. The associated string is the one that would be returned if the long value were passed to [Long.toString(long)](http://docs.google.com/java/lang/Long.html#toString(long)). This method is intended for use in conjunction with [getLong(java.lang.String, long)](http://docs.google.com/java/util/prefs/Preferences.html#getLong(java.lang.String,%20long)).

**Parameters:**key - key with which the string form of value is to be associated.value - value whose string form is to be associated with key. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if key.length() exceeds MAX\_KEY\_LENGTH. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[getLong(String,long)](http://docs.google.com/java/util/prefs/Preferences.html#getLong(java.lang.String,%20long))

### getLong

public abstract long **getLong**([String](http://docs.google.com/java/lang/String.html) key,  
 long def)

Returns the long value represented by the string associated with the specified key in this preference node. The string is converted to a long as by [Long.parseLong(String)](http://docs.google.com/java/lang/Long.html#parseLong(java.lang.String)). Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if Long.parseLong(String) would throw a [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html) if the associated value were passed. This method is intended for use in conjunction with [putLong(java.lang.String, long)](http://docs.google.com/java/util/prefs/Preferences.html#putLong(java.lang.String,%20long)).

If the implementation supports *stored defaults* and such a default exists, is accessible, and could be converted to a long with Long.parseLong, this long is returned in preference to the specified default.

**Parameters:**key - key whose associated value is to be returned as a long.def - the value to be returned in the event that this preference node has no value associated with key or the associated value cannot be interpreted as a long, or the backing store is inaccessible. **Returns:**the long value represented by the string associated with key in this preference node, or def if the associated value does not exist or cannot be interpreted as a long. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null.**See Also:**[putLong(String,long)](http://docs.google.com/java/util/prefs/Preferences.html#putLong(java.lang.String,%20long)), [get(String,String)](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String))

### putBoolean

public abstract void **putBoolean**([String](http://docs.google.com/java/lang/String.html) key,  
 boolean value)

Associates a string representing the specified boolean value with the specified key in this preference node. The associated string is "true" if the value is true, and "false" if it is false. This method is intended for use in conjunction with [getBoolean(java.lang.String, boolean)](http://docs.google.com/java/util/prefs/Preferences.html#getBoolean(java.lang.String,%20boolean)).

**Parameters:**key - key with which the string form of value is to be associated.value - value whose string form is to be associated with key. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if key.length() exceeds MAX\_KEY\_LENGTH. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[getBoolean(String,boolean)](http://docs.google.com/java/util/prefs/Preferences.html#getBoolean(java.lang.String,%20boolean)), [get(String,String)](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String))

### getBoolean

public abstract boolean **getBoolean**([String](http://docs.google.com/java/lang/String.html) key,  
 boolean def)

Returns the boolean value represented by the string associated with the specified key in this preference node. Valid strings are "true", which represents true, and "false", which represents false. Case is ignored, so, for example, "TRUE" and "False" are also valid. This method is intended for use in conjunction with [putBoolean(java.lang.String, boolean)](http://docs.google.com/java/util/prefs/Preferences.html#putBoolean(java.lang.String,%20boolean)).

Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if the associated value is something other than "true" or "false", ignoring case.

If the implementation supports *stored defaults* and such a default exists and is accessible, it is used in preference to the specified default, unless the stored default is something other than "true" or "false", ignoring case, in which case the specified default is used.

**Parameters:**key - key whose associated value is to be returned as a boolean.def - the value to be returned in the event that this preference node has no value associated with key or the associated value cannot be interpreted as a boolean, or the backing store is inaccessible. **Returns:**the boolean value represented by the string associated with key in this preference node, or def if the associated value does not exist or cannot be interpreted as a boolean. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null.**See Also:**[get(String,String)](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String)), [putBoolean(String,boolean)](http://docs.google.com/java/util/prefs/Preferences.html#putBoolean(java.lang.String,%20boolean))

### putFloat

public abstract void **putFloat**([String](http://docs.google.com/java/lang/String.html) key,  
 float value)

Associates a string representing the specified float value with the specified key in this preference node. The associated string is the one that would be returned if the float value were passed to [Float.toString(float)](http://docs.google.com/java/lang/Float.html#toString(float)). This method is intended for use in conjunction with [getFloat(java.lang.String, float)](http://docs.google.com/java/util/prefs/Preferences.html#getFloat(java.lang.String,%20float)).

**Parameters:**key - key with which the string form of value is to be associated.value - value whose string form is to be associated with key. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if key.length() exceeds MAX\_KEY\_LENGTH. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[getFloat(String,float)](http://docs.google.com/java/util/prefs/Preferences.html#getFloat(java.lang.String,%20float))

### getFloat

public abstract float **getFloat**([String](http://docs.google.com/java/lang/String.html) key,  
 float def)

Returns the float value represented by the string associated with the specified key in this preference node. The string is converted to an integer as by [Float.parseFloat(String)](http://docs.google.com/java/lang/Float.html#parseFloat(java.lang.String)). Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if Float.parseFloat(String) would throw a [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html) if the associated value were passed. This method is intended for use in conjunction with [putFloat(java.lang.String, float)](http://docs.google.com/java/util/prefs/Preferences.html#putFloat(java.lang.String,%20float)).

If the implementation supports *stored defaults* and such a default exists, is accessible, and could be converted to a float with Float.parseFloat, this float is returned in preference to the specified default.

**Parameters:**key - key whose associated value is to be returned as a float.def - the value to be returned in the event that this preference node has no value associated with key or the associated value cannot be interpreted as a float, or the backing store is inaccessible. **Returns:**the float value represented by the string associated with key in this preference node, or def if the associated value does not exist or cannot be interpreted as a float. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null.**See Also:**[putFloat(String,float)](http://docs.google.com/java/util/prefs/Preferences.html#putFloat(java.lang.String,%20float)), [get(String,String)](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String))

### putDouble

public abstract void **putDouble**([String](http://docs.google.com/java/lang/String.html) key,  
 double value)

Associates a string representing the specified double value with the specified key in this preference node. The associated string is the one that would be returned if the double value were passed to [Double.toString(double)](http://docs.google.com/java/lang/Double.html#toString(double)). This method is intended for use in conjunction with [getDouble(java.lang.String, double)](http://docs.google.com/java/util/prefs/Preferences.html#getDouble(java.lang.String,%20double)).

**Parameters:**key - key with which the string form of value is to be associated.value - value whose string form is to be associated with key. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if key.length() exceeds MAX\_KEY\_LENGTH. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[getDouble(String,double)](http://docs.google.com/java/util/prefs/Preferences.html#getDouble(java.lang.String,%20double))

### getDouble

public abstract double **getDouble**([String](http://docs.google.com/java/lang/String.html) key,  
 double def)

Returns the double value represented by the string associated with the specified key in this preference node. The string is converted to an integer as by [Double.parseDouble(String)](http://docs.google.com/java/lang/Double.html#parseDouble(java.lang.String)). Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if Double.parseDouble(String) would throw a [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html) if the associated value were passed. This method is intended for use in conjunction with [putDouble(java.lang.String, double)](http://docs.google.com/java/util/prefs/Preferences.html#putDouble(java.lang.String,%20double)).

If the implementation supports *stored defaults* and such a default exists, is accessible, and could be converted to a double with Double.parseDouble, this double is returned in preference to the specified default.

**Parameters:**key - key whose associated value is to be returned as a double.def - the value to be returned in the event that this preference node has no value associated with key or the associated value cannot be interpreted as a double, or the backing store is inaccessible. **Returns:**the double value represented by the string associated with key in this preference node, or def if the associated value does not exist or cannot be interpreted as a double. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null.**See Also:**[putDouble(String,double)](http://docs.google.com/java/util/prefs/Preferences.html#putDouble(java.lang.String,%20double)), [get(String,String)](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String))

### putByteArray

public abstract void **putByteArray**([String](http://docs.google.com/java/lang/String.html) key,  
 byte[] value)

Associates a string representing the specified byte array with the specified key in this preference node. The associated string is the *Base64* encoding of the byte array, as defined in [RFC 2045](http://www.ietf.org/rfc/rfc2045.txt), Section 6.8, with one minor change: the string will consist solely of characters from the *Base64 Alphabet*; it will not contain any newline characters. Note that the maximum length of the byte array is limited to three quarters of MAX\_VALUE\_LENGTH so that the length of the Base64 encoded String does not exceed MAX\_VALUE\_LENGTH. This method is intended for use in conjunction with [getByteArray(java.lang.String, byte[])](http://docs.google.com/java/util/prefs/Preferences.html#getByteArray(java.lang.String,%20byte%5B%5D)).

**Parameters:**key - key with which the string form of value is to be associated.value - value whose string form is to be associated with key. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key or value is null. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if key.length() exceeds MAX\_KEY\_LENGTH or if value.length exceeds MAX\_VALUE\_LENGTH\*3/4. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[getByteArray(String,byte[])](http://docs.google.com/java/util/prefs/Preferences.html#getByteArray(java.lang.String,%20byte%5B%5D)), [get(String,String)](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String))

### getByteArray

public abstract byte[] **getByteArray**([String](http://docs.google.com/java/lang/String.html) key,  
 byte[] def)

Returns the byte array value represented by the string associated with the specified key in this preference node. Valid strings are *Base64* encoded binary data, as defined in [RFC 2045](http://www.ietf.org/rfc/rfc2045.txt), Section 6.8, with one minor change: the string must consist solely of characters from the *Base64 Alphabet*; no newline characters or extraneous characters are permitted. This method is intended for use in conjunction with [putByteArray(java.lang.String, byte[])](http://docs.google.com/java/util/prefs/Preferences.html#putByteArray(java.lang.String,%20byte%5B%5D)).

Returns the specified default if there is no value associated with the key, the backing store is inaccessible, or if the associated value is not a valid Base64 encoded byte array (as defined above).

If the implementation supports *stored defaults* and such a default exists and is accessible, it is used in preference to the specified default, unless the stored default is not a valid Base64 encoded byte array (as defined above), in which case the specified default is used.

**Parameters:**key - key whose associated value is to be returned as a byte array.def - the value to be returned in the event that this preference node has no value associated with key or the associated value cannot be interpreted as a byte array, or the backing store is inaccessible. **Returns:**the byte array value represented by the string associated with key in this preference node, or def if the associated value does not exist or cannot be interpreted as a byte array. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if key is null. (A null value for def *is* permitted.)**See Also:**[get(String,String)](http://docs.google.com/java/util/prefs/Preferences.html#get(java.lang.String,%20java.lang.String)), [putByteArray(String,byte[])](http://docs.google.com/java/util/prefs/Preferences.html#putByteArray(java.lang.String,%20byte%5B%5D))

### keys

public abstract [String](http://docs.google.com/java/lang/String.html)[] **keys**()  
 throws [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Returns all of the keys that have an associated value in this preference node. (The returned array will be of size zero if this node has no preferences.)

If the implementation supports *stored defaults* and there are any such defaults at this node that have not been overridden, by explicit preferences, the defaults are returned in the array in addition to any explicit preferences.

**Returns:**an array of the keys that have an associated value in this preference node. **Throws:** [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.

### childrenNames

public abstract [String](http://docs.google.com/java/lang/String.html)[] **childrenNames**()  
 throws [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Returns the names of the children of this preference node, relative to this node. (The returned array will be of size zero if this node has no children.)

**Returns:**the names of the children of this preference node. **Throws:** [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.

### parent

public abstract [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) **parent**()

Returns the parent of this preference node, or null if this is the root.

**Returns:**the parent of this preference node. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.

### node

public abstract [Preferences](http://docs.google.com/java/util/prefs/Preferences.html) **node**([String](http://docs.google.com/java/lang/String.html) pathName)

Returns the named preference node in the same tree as this node, creating it and any of its ancestors if they do not already exist. Accepts a relative or absolute path name. Relative path names (which do not begin with the slash character ('/')) are interpreted relative to this preference node.

If the returned node did not exist prior to this call, this node and any ancestors that were created by this call are not guaranteed to become permanent until the flush method is called on the returned node (or one of its ancestors or descendants).

**Parameters:**pathName - the path name of the preference node to return. **Returns:**the specified preference node. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the path name is invalid (i.e., it contains multiple consecutive slash characters, or ends with a slash character and is more than one character long). [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if path name is null. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[flush()](http://docs.google.com/java/util/prefs/Preferences.html#flush())

### nodeExists

public abstract boolean **nodeExists**([String](http://docs.google.com/java/lang/String.html) pathName)  
 throws [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Returns true if the named preference node exists in the same tree as this node. Relative path names (which do not begin with the slash character ('/')) are interpreted relative to this preference node.

If this node (or an ancestor) has already been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method, it *is* legal to invoke this method, but only with the path name ""; the invocation will return false. Thus, the idiom p.nodeExists("") may be used to test whether p has been removed.

**Parameters:**pathName - the path name of the node whose existence is to be checked. **Returns:**true if the specified node exists. **Throws:** [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the path name is invalid (i.e., it contains multiple consecutive slash characters, or ends with a slash character and is more than one character long). [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if path name is null. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method and pathName is not the empty string ("").

### removeNode

public abstract void **removeNode**()  
 throws [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Removes this preference node and all of its descendants, invalidating any preferences contained in the removed nodes. Once a node has been removed, attempting any method other than [name()](http://docs.google.com/java/util/prefs/Preferences.html#name()), [absolutePath()](http://docs.google.com/java/util/prefs/Preferences.html#absolutePath()), [isUserNode()](http://docs.google.com/java/util/prefs/Preferences.html#isUserNode()), [flush()](http://docs.google.com/java/util/prefs/Preferences.html#flush()) or [nodeExists("")](http://docs.google.com/java/util/prefs/Preferences.html#node(java.lang.String)) on the corresponding Preferences instance will fail with an IllegalStateException. (The methods defined on [Object](http://docs.google.com/java/lang/Object.html) can still be invoked on a node after it has been removed; they will not throw IllegalStateException.)

The removal is not guaranteed to be persistent until the flush method is called on this node (or an ancestor).

If this implementation supports *stored defaults*, removing a node exposes any stored defaults at or below this node. Thus, a subsequent call to nodeExists on this node's path name may return true, and a subsequent call to node on this path name may return a (different) Preferences instance representing a non-empty collection of preferences and/or children.

**Throws:** [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has already been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method. [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this method is invoked on the root node.**See Also:**[flush()](http://docs.google.com/java/util/prefs/Preferences.html#flush())

### name

public abstract [String](http://docs.google.com/java/lang/String.html) **name**()

Returns this preference node's name, relative to its parent.

**Returns:**this preference node's name, relative to its parent.

### absolutePath

public abstract [String](http://docs.google.com/java/lang/String.html) **absolutePath**()

Returns this preference node's absolute path name.

**Returns:**this preference node's absolute path name.

### isUserNode

public abstract boolean **isUserNode**()

Returns true if this preference node is in the user preference tree, false if it's in the system preference tree.

**Returns:**true if this preference node is in the user preference tree, false if it's in the system preference tree.

### toString

public abstract [String](http://docs.google.com/java/lang/String.html) **toString**()

Returns a string representation of this preferences node, as if computed by the expression:(this.isUserNode() ? "User" : "System") + " Preference Node: " + this.absolutePath().

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a string representation of the object.

### flush

public abstract void **flush**()  
 throws [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Forces any changes in the contents of this preference node and its descendants to the persistent store. Once this method returns successfully, it is safe to assume that all changes made in the subtree rooted at this node prior to the method invocation have become permanent.

Implementations are free to flush changes into the persistent store at any time. They do not need to wait for this method to be called.

When a flush occurs on a newly created node, it is made persistent, as are any ancestors (and descendants) that have yet to be made persistent. Note however that any preference value changes in ancestors are *not* guaranteed to be made persistent.

If this method is invoked on a node that has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method, flushSpi() is invoked on this node, but not on others.

**Throws:** [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it.**See Also:**[sync()](http://docs.google.com/java/util/prefs/Preferences.html#sync())

### sync

public abstract void **sync**()  
 throws [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Ensures that future reads from this preference node and its descendants reflect any changes that were committed to the persistent store (from any VM) prior to the sync invocation. As a side-effect, forces any changes in the contents of this preference node and its descendants to the persistent store, as if the flush method had been invoked on this node.

**Throws:** [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if this operation cannot be completed due to a failure in the backing store, or inability to communicate with it. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[flush()](http://docs.google.com/java/util/prefs/Preferences.html#flush())

### addPreferenceChangeListener

public abstract void **addPreferenceChangeListener**([PreferenceChangeListener](http://docs.google.com/java/util/prefs/PreferenceChangeListener.html) pcl)

Registers the specified listener to receive *preference change events* for this preference node. A preference change event is generated when a preference is added to this node, removed from this node, or when the value associated with a preference is changed. (Preference change events are *not* generated by the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method, which generates a *node change event*. Preference change events *are* generated by the clear method.)

Events are only guaranteed for changes made within the same JVM as the registered listener, though some implementations may generate events for changes made outside this JVM. Events may be generated before the changes have been made persistent. Events are not generated when preferences are modified in descendants of this node; a caller desiring such events must register with each descendant.

**Parameters:**pcl - The preference change listener to add. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if pcl is null. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[removePreferenceChangeListener(PreferenceChangeListener)](http://docs.google.com/java/util/prefs/Preferences.html#removePreferenceChangeListener(java.util.prefs.PreferenceChangeListener)), [addNodeChangeListener(NodeChangeListener)](http://docs.google.com/java/util/prefs/Preferences.html#addNodeChangeListener(java.util.prefs.NodeChangeListener))

### removePreferenceChangeListener

public abstract void **removePreferenceChangeListener**([PreferenceChangeListener](http://docs.google.com/java/util/prefs/PreferenceChangeListener.html) pcl)

Removes the specified preference change listener, so it no longer receives preference change events.

**Parameters:**pcl - The preference change listener to remove. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if pcl was not a registered preference change listener on this node. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[addPreferenceChangeListener(PreferenceChangeListener)](http://docs.google.com/java/util/prefs/Preferences.html#addPreferenceChangeListener(java.util.prefs.PreferenceChangeListener))

### addNodeChangeListener

public abstract void **addNodeChangeListener**([NodeChangeListener](http://docs.google.com/java/util/prefs/NodeChangeListener.html) ncl)

Registers the specified listener to receive *node change events* for this node. A node change event is generated when a child node is added to or removed from this node. (A single [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) invocation results in multiple *node change events*, one for every node in the subtree rooted at the removed node.)

Events are only guaranteed for changes made within the same JVM as the registered listener, though some implementations may generate events for changes made outside this JVM. Events may be generated before the changes have become permanent. Events are not generated when indirect descendants of this node are added or removed; a caller desiring such events must register with each descendant.

Few guarantees can be made regarding node creation. Because nodes are created implicitly upon access, it may not be feasible for an implementation to determine whether a child node existed in the backing store prior to access (for example, because the backing store is unreachable or cached information is out of date). Under these circumstances, implementations are neither required to generate node change events nor prohibited from doing so.

**Parameters:**ncl - The NodeChangeListener to add. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if ncl is null. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[removeNodeChangeListener(NodeChangeListener)](http://docs.google.com/java/util/prefs/Preferences.html#removeNodeChangeListener(java.util.prefs.NodeChangeListener)), [addPreferenceChangeListener(PreferenceChangeListener)](http://docs.google.com/java/util/prefs/Preferences.html#addPreferenceChangeListener(java.util.prefs.PreferenceChangeListener))

### removeNodeChangeListener

public abstract void **removeNodeChangeListener**([NodeChangeListener](http://docs.google.com/java/util/prefs/NodeChangeListener.html) ncl)

Removes the specified NodeChangeListener, so it no longer receives change events.

**Parameters:**ncl - The NodeChangeListener to remove. **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if ncl was not a registered NodeChangeListener on this node. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[addNodeChangeListener(NodeChangeListener)](http://docs.google.com/java/util/prefs/Preferences.html#addNodeChangeListener(java.util.prefs.NodeChangeListener))

### exportNode

public abstract void **exportNode**([OutputStream](http://docs.google.com/java/io/OutputStream.html) os)  
 throws [IOException](http://docs.google.com/java/io/IOException.html),  
 [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Emits on the specified output stream an XML document representing all of the preferences contained in this node (but not its descendants). This XML document is, in effect, an offline backup of the node.

The XML document will have the following DOCTYPE declaration:

<!DOCTYPE preferences SYSTEM "http://java.sun.com/dtd/preferences.dtd">

The UTF-8 character encoding will be used.

This method is an exception to the general rule that the results of concurrently executing multiple methods in this class yields results equivalent to some serial execution. If the preferences at this node are modified concurrently with an invocation of this method, the exported preferences comprise a "fuzzy snapshot" of the preferences contained in the node; some of the concurrent modifications may be reflected in the exported data while others may not.

**Parameters:**os - the output stream on which to emit the XML document. **Throws:** [IOException](http://docs.google.com/java/io/IOException.html) - if writing to the specified output stream results in an IOException. [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if preference data cannot be read from backing store. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[importPreferences(InputStream)](http://docs.google.com/java/util/prefs/Preferences.html#importPreferences(java.io.InputStream))

### exportSubtree

public abstract void **exportSubtree**([OutputStream](http://docs.google.com/java/io/OutputStream.html) os)  
 throws [IOException](http://docs.google.com/java/io/IOException.html),  
 [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html)

Emits an XML document representing all of the preferences contained in this node and all of its descendants. This XML document is, in effect, an offline backup of the subtree rooted at the node.

The XML document will have the following DOCTYPE declaration:

<!DOCTYPE preferences SYSTEM "http://java.sun.com/dtd/preferences.dtd">

The UTF-8 character encoding will be used.

This method is an exception to the general rule that the results of concurrently executing multiple methods in this class yields results equivalent to some serial execution. If the preferences or nodes in the subtree rooted at this node are modified concurrently with an invocation of this method, the exported preferences comprise a "fuzzy snapshot" of the subtree; some of the concurrent modifications may be reflected in the exported data while others may not.

**Parameters:**os - the output stream on which to emit the XML document. **Throws:** [IOException](http://docs.google.com/java/io/IOException.html) - if writing to the specified output stream results in an IOException. [BackingStoreException](http://docs.google.com/java/util/prefs/BackingStoreException.html) - if preference data cannot be read from backing store. [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this node (or an ancestor) has been removed with the [removeNode()](http://docs.google.com/java/util/prefs/Preferences.html#removeNode()) method.**See Also:**[importPreferences(InputStream)](http://docs.google.com/java/util/prefs/Preferences.html#importPreferences(java.io.InputStream)), [exportNode(OutputStream)](http://docs.google.com/java/util/prefs/Preferences.html#exportNode(java.io.OutputStream))

### importPreferences

public static void **importPreferences**([InputStream](http://docs.google.com/java/io/InputStream.html) is)  
 throws [IOException](http://docs.google.com/java/io/IOException.html),  
 [InvalidPreferencesFormatException](http://docs.google.com/java/util/prefs/InvalidPreferencesFormatException.html)

Imports all of the preferences represented by the XML document on the specified input stream. The document may represent user preferences or system preferences. If it represents user preferences, the preferences will be imported into the calling user's preference tree (even if they originally came from a different user's preference tree). If any of the preferences described by the document inhabit preference nodes that do not exist, the nodes will be created.

The XML document must have the following DOCTYPE declaration:

<!DOCTYPE preferences SYSTEM "http://java.sun.com/dtd/preferences.dtd">

(This method is designed for use in conjunction with [exportNode(OutputStream)](http://docs.google.com/java/util/prefs/Preferences.html#exportNode(java.io.OutputStream)) and [exportSubtree(OutputStream)](http://docs.google.com/java/util/prefs/Preferences.html#exportSubtree(java.io.OutputStream)).

This method is an exception to the general rule that the results of concurrently executing multiple methods in this class yields results equivalent to some serial execution. The method behaves as if implemented on top of the other public methods in this class, notably [node(String)](http://docs.google.com/java/util/prefs/Preferences.html#node(java.lang.String)) and [put(String, String)](http://docs.google.com/java/util/prefs/Preferences.html#put(java.lang.String,%20java.lang.String)).

**Parameters:**is - the input stream from which to read the XML document. **Throws:** [IOException](http://docs.google.com/java/io/IOException.html) - if reading from the specified input stream results in an IOException. [InvalidPreferencesFormatException](http://docs.google.com/java/util/prefs/InvalidPreferencesFormatException.html) - Data on input stream does not constitute a valid XML document with the mandated document type. [SecurityException](http://docs.google.com/java/lang/SecurityException.html) - If a security manager is present and it denies RuntimePermission("preferences").**See Also:**[RuntimePermission](http://docs.google.com/java/lang/RuntimePermission.html)

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Preferences.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/java/util/prefs/PreferenceChangeListener.html)   [**NEXT CLASS**](http://docs.google.com/java/util/prefs/PreferencesFactory.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/prefs/Preferences.html)    [**NO FRAMES**](http://docs.google.com/Preferences.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#3rdcrjn) | [METHOD](#lnxbz9) |

[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).