

# Java Platform, Standard Edition

## Internationalization Guide



Release 23  
F99988-01  
September 2024



Copyright © 1993, 2024, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

# Contents

## Preface

---

|                             |    |
|-----------------------------|----|
| Audience                    | vi |
| Documentation Accessibility | vi |
| Diversity and Inclusion     | vi |
| Related Documents           | vi |
| Conventions                 | vi |

## 1 Internationalization Overview

---

|  |     |
|--|-----|
| Text Representation                    | 1-1 |
| Locale Identification and Localization | 1-2 |
| Locales                                | 1-2 |
| Locale Class                           | 1-2 |
| Supported Locales                      | 1-2 |
| Localized Resources                    | 1-2 |
| ResourceBundle Class                   | 1-3 |
| ListResourceBundle Class               | 1-3 |
| PropertyResourceBundle Class           | 1-3 |
| CLDR Locale Data Enabled by Default    | 1-3 |
| Date and Time Handling                 | 1-4 |
| Text Processing                        | 1-4 |
| Formatting                             | 1-4 |
| Format Class                           | 1-4 |
| DateFormat Class                       | 1-5 |
| SimpleDateFormat Class                 | 1-5 |
| DateFormatSymbols Class                | 1-5 |
| NumberFormat Class                     | 1-5 |
| DecimalFormat Class                    | 1-6 |
| DecimalFormatSymbols Class             | 1-6 |
| ChoiceFormat Class                     | 1-6 |
| MessageFormat Class                    | 1-6 |
| ParsePosition Class                    | 1-7 |
| FieldPosition Class                    | 1-7 |
| Locale-Sensitive String Operations     | 1-7 |

|                                |      |
|--------------------------------|------|
| Collator Class                 | 1-7  |
| RuleBasedCollator Class        | 1-7  |
| CollationElementIterator Class | 1-7  |
| CollationKey Class             | 1-7  |
| BreakIterator Class            | 1-8  |
| StringCharacterIterator Class  | 1-8  |
| CharacterIterator Interface    | 1-8  |
| Normalizer Class               | 1-8  |
| Locale-Sensitive Services SPIs | 1-8  |
| Character Encoding Conversion  | 1-9  |
| Supported Encodings            | 1-9  |
| Stream I/O                     | 1-9  |
| Reader and Writer Classes      | 1-9  |
| PrintStream Class              | 1-9  |
| Charset Package                | 1-9  |
| Input Methods                  | 1-10 |
| Input Method Support in Swing  | 1-10 |
| Input Method Framework         | 1-10 |

## 2 Supported Encodings

---

|   |      |
|---|------|
| Basic Encoding Set (contained in java.base module)                                  | 2-1  |
| Extended Encoding Set (contained in jdk.charsets module)                            | 2-4  |
| Printing Charset Information  | 2-11 |
| Default Charset   | 2-11 |
| Default Charset for JDK 17 and Earlier Releases                                     | 2-12 |
| Changing the JDK's Default Charset  | 2-12 |
| Running Java Applications on JDK Whose Default Charset Is Determined by Environment | 2-13 |
| Ensuring Source File Encoding Is Compatible with Your JDK                           | 2-13 |

## 3 Supported Calendars

---

## 4 Supported Fonts

---

|                            |     |
|----------------------------|-----|
| Support for Physical Fonts | 4-1 |
| Support for Logical Fonts  | 4-1 |

## 5 Font Configuration Files

---

|                                  |     |
|----------------------------------|-----|
| Supported Platforms              | 5-1 |
| Loading Font Configuration Files | 5-1 |

|  |     |
|--|-----|
| Names Used in Font Configuration Files | 5-2 |
| Properties for All Platforms           | 5-3 |
| Version Property                       | 5-3 |
| Component Font Mappings                | 5-3 |
| Search Sequences                       | 5-3 |
| Exclusion Ranges                       | 5-5 |
| Proportional Fonts                     | 5-5 |
| Font File Names                        | 5-6 |
| Appended Font Path                     | 5-6 |
| Properties for Windows                 | 5-7 |
| Property for Linux                     | 5-7 |

# Preface

This guide summarizes the internationalization APIs and features of the Java SE Platform.

## Audience

This guide is intended for Java programmers who want to design applications so that they can be adapted to various languages and regions without engineering changes.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

## Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

## Related Documents

For coding examples and step-by-step instructions, see the [Internationalization Trail](#) in The Java Tutorials (Java SE 8 and earlier).

## Conventions

The following text conventions are used in this document:

| Convention      | Meaning  |
|-----------------|--|
| <b>boldface</b> | Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary. |

| <b>Convention</b> | <b>Meaning</b>   |
|-------------------|--|
| <i>italic</i>     | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.                          |
| monospace         | Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter. |

---

# 1

## Internationalization Overview

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. Sometimes the term internationalization is abbreviated as i18n, because there are 18 letters between the first "i" and the last "n."

An internationalized program has the following characteristics:

- With the addition of localization data, the same executable can run worldwide.
- Textual elements, such as status messages and the GUI component labels, are not hardcoded in the program. Instead they are stored outside the source code and retrieved dynamically.
- Support for new languages does not require recompilation.
- Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
- It can be localized quickly.

The internet demands global software - that is, software that can be developed independently of the countries or languages of its users, and then localized for multiple countries or regions. The Java Platform provides a rich set of APIs for developing global applications. These internationalization APIs are based on the Unicode standard and include the ability to adapt text, numbers, dates, currency, and user-defined objects to any country's conventions.

This guide summarizes the internationalization APIs and features of the Java Platform, Standard Edition. For coding examples and step-by-step instructions, see the [Internationalization Trail](#) in the Java Tutorials.

## Text Representation

The Java programming language is based on the Unicode character set, and several libraries implement the Unicode standard. Unicode is an international character set standard which supports all of the major scripts of the world, as well as common technical symbols. The original Unicode specification defined characters as fixed-width 16-bit entities, but the Unicode standard has since been changed to allow for characters whose representation requires more than 16 bits. The range of legal code points is now U+0000 to U+10FFFF. An encoding defined by the standard, UTF-16, allows to represent all Unicode code points using one or two 16-bit units.

The primitive data type `char` in the Java programming language is an unsigned 16-bit integer that can represent a Unicode code point in the range U+0000 to U+FFFF, or the code units of UTF-16. The various types and classes in the Java platform that represent character sequences - `char[]`, implementations of `java.lang.CharSequence` (such as the `String` class), and implementations of `java.text.CharacterIterator` - are UTF-16 sequences. Most Java source code is written in ASCII, a 7-bit character encoding, or ISO-8859-1, an 8-bit character encoding, but is translated into UTF-16 before processing.

The `Character` class is an object wrapper for the `char` primitive type. The `Character` class also contains static methods such as `isLowerCase()` and `isDigit()` for determining the properties of a character. These methods have overloads that accept either a `char` (which



allows representation of Unicode code points in the range U+0000 to U+FFFF) or an int (which allows representation of all Unicode code points).

## Locale Identification and Localization

A `Locale` object is an identifier for a particular combination of language and region. Localization is the process of adapting software for a specific region or language by adding locale-specific components and translating text.

### Locales

On the Java platform, a locale is simply an identifier for a particular combination of language and region. It is not a collection of locale-specific attributes. Instead, each locale-sensitive class maintains its own locale-specific information. With this design, there is no difference in how user and system objects maintain their locale-specific resources. Both use the standard localization mechanism.

Java programs are *not* assigned a single global locale. All locale-sensitive operations may be explicitly given a locale as an argument. This greatly simplifies multilingual programs. While a global locale is not enforced, a default locale is available for programs that do not wish to manage locales explicitly. A default locale also makes it possible to affect the behavior of the entire presentation with a single choice.

Java locales act as requests for certain behavior from another object. For example, a French Canadian locale passed to a `Calendar` object asks that the `Calendar` behave correctly for the customs of Quebec. It is up to the object accepting the locale to do the right thing. If the object has not been localized for a particular locale, it will try to find a "close" match with a locale for which it has been localized. Thus if a `Calendar` object was not localized for French Canada, but was localized for the French language in general, it would use the French localization instead.

### Locale Class

A `Locale` object represents a specific geographical, political, or cultural region. An operation that requires a locale to perform its task is called locale-sensitive and uses the `Locale` object to tailor information for the user. For example, displaying a number is a locale-sensitive operation - the number should be formatted according to the customs and conventions of the user's native country, region, or culture.

### Supported Locales

On the Java platform, there does not have to be a single set of supported locales, since each class maintains its own localizations. Nevertheless, there is a consistent set of localizations supported by the classes of the Java Platform. Other implementations of the Java Platform may support different locales. Locales that are supported by the JDK are summarized by release. See [JDK 23 Supported Locales](#) for what is supported.

### Localized Resources

All locale-sensitive classes must be able to access resources customized for the locales they support. To aid in the process of localization, it helps to have these resources grouped together by locale and separated from the locale-neutral parts of the program.

## ResourceBundle Class

The class `ResourceBundle` is an abstract base class representing containers of resources. Programmers create subclasses of `ResourceBundle` that contain resources for a particular locale. New resources can be added to an instance of `ResourceBundle`, or new instances of `ResourceBundle` can be added to a system without affecting the code that uses them. Packaging resources as classes allows developers to take advantage of Java's class loading mechanism to find resources.

Resource bundles contain locale-specific objects. When a program needs a locale-specific resource, such as a `String` object, the program can load it from the resource bundle that is appropriate for the current user's locale. In this way, the programmer can write code that is largely independent of the user's locale, isolating most, if not all, of the locale-specific information in resource bundles.

This allows Java programmers to write code that can

- be easily localized, or translated, into different languages
- handle multiple locales at once
- be easily modified later to support even more locales

## ResourceBundle.Control Class

`ResourceBundle.Control` is a nested class of `ResourceBundle`. It defines methods to be called by the `ResourceBundle.getBundle` factory methods so that the resource bundle loading behavior may be changed. For example, application specific resource bundle formats, such as XML, could be supported by overriding the methods.

`ResourceBundle.Control` is not supported in named modules. Existing code using `Control` is expected to work, but for new code in a named module, implement `basenameProvider` and load the resource bundle from there. See [Resource Bundles and Named Modules](#).

## ListResourceBundle Class

`ListResourceBundle` is an abstract subclass of `ResourceBundle` that manages resources for a locale in a convenient and easy to use list.

## PropertyResourceBundle Class

`PropertyResourceBundle` is a concrete subclass of `ResourceBundle` that manages resources for a locale using a set of static strings from a property file.

## CLDR Locale Data Enabled by Default

The XML-based locale data of the Unicode Common Locale Data Repository (CLDR), first added in JDK 8, is the default locale data since JDK 9.

There are three distinct sources for locale data, identified by the following keywords:

- `CLDR` represents the locale data provided by the Unicode [CLDR](#) project.
- `HOST` represents the current user's customization of the underlying operating system's settings. It works only with the user's default locale, and the customizable settings may

vary depending on the operating system. However, primarily date, time, number, and currency formats are supported.

- SPI represents the locale-sensitive services implemented by the installed Service Provider Interface (SPI) providers.

To select a locale data source, use the `java.locale.providers` system property, listing the data sources in the preferred order. If a provider cannot offer the requested locale data, the search proceeds to the next provider in order. For example:

```
java.locale.providers=HOST,SPI,CLDR
```

If you do not set this property, the default behavior is equivalent to the following setting:

```
java.locale.providers=CLDR
```

See [JEP 252: Use CLDR Locale Data by Default](#) for more detail.

For supported locales, see [JDK 23 Supported Locales](#). See [java.util.api.LocaleServiceProvider](#) for the related API.

## Date and Time Handling

The Date-Time package, `java.time`, provides a comprehensive model for date and time. Although `java.time` is based on the International Organization for Standardization (ISO) calendar system, commonly used global calendars are also supported.

See [The Date-Time Packages](#) lesson in The Java Tutorials (Java SE 8 and earlier).

## Text Processing

Text processing involves formatting locale-sensitive information such as, currencies, dates, times, and text messages. It also includes manipulating text in a locale-sensitive manner, meaning that string operations, such as searching and sorting, are properly performed regardless of locale.

### Formatting

It is in formatting data for output that many cultural conventions are applied. Numbers, dates, times, and messages may all require formatting before they can be displayed. The Java platform provides a set of flexible formatting classes that can handle both the standard locale formats and programmer defined custom formats. These formatting classes are also able to parse formatted strings back into their constituent objects.

### Format Class

The class `Format` is an abstract base class for formatting locale-sensitive information such as dates, times, messages, and numbers. Three main subclasses are provided: `DateFormat`, `NumberFormat`, and `MessageFormat`. These three also provide subclasses of their own.

## DateFormat Class

Dates and times are stored internally in a locale-independent way, but should be formatted so that they can be displayed in a locale-sensitive manner. For example, the same date might be formatted as follows:

- November 3, 1997 (English)
- 3 novembre 1997 (French)

The class `DateFormat` is an abstract base class for formatting and parsing date and time values in a locale-independent manner. It has a number of static factory methods for getting standard time formats for a given locale.

The `DateFormat` object uses `Calendar` and `TimeZone` objects in order to interpret time values. By default, a `DateFormat` object for a given locale will use the appropriate `Calendar` object for that locale and the system's default `TimeZone` object. The programmer can override these choices if desired.

## SimpleDateFormat Class

The class `SimpleDateFormat` is a concrete class for formatting and parsing dates and times in a locale-sensitive manner. It allows for formatting (milliseconds to text), parsing (text to milliseconds), and normalization.

## DateFormatSymbols Class

The class `DateFormatSymbols` is used to encapsulate localizable date-time formatting data, such as the names of the months, the names of the days of the week, time of day, and the time zone data. The `DateFormat` and `SimpleDateFormat` classes both use the `DateFormatSymbols` class to encapsulate this information.

Usually, programmers will not use the `DateFormatSymbols` directly. Rather, they will implement formatting with the `DateFormat` class's factory methods.

## NumberFormat Class

The class `NumberFormat` is an abstract base class for formatting and parsing numeric data. It contains a number of static factory methods for getting different kinds of locale-specific number formats.

The `NumberFormat` class helps programmers to format and parse numbers for any locale. Code using this class can be completely independent of the locale conventions for decimal points, thousands-separators, the particular decimal digits used, or whether the number format is even decimal. The application can also display a number as a normal decimal number, currency, or percentage:

- 1,234.5 (decimal number in U.S. format)
- \$1,234.50 (U.S. currency in U.S. format)
- 1.234,50 € (European currency in German format)
- 123.450% (percent in German format)

## DecimalFormat Class

Numbers are stored internally in a locale-independent way, but should be formatted so that they can be displayed in a locale-sensitive manner. For example, when using "#,###.00" as a pattern, the same number might be formatted as follows:

- 1.234,56 (German)
- 1,234.56 (English)

The class `DecimalFormat`, which is a concrete subclass of the `NumberFormat` class, can format decimal numbers. Programmers generally will not instantiate this class directly but will use the factory methods provided.

The `DecimalFormat` class has the ability to take a pattern string to specify how a number should be formatted. The pattern specifies attributes such as the precision of the number, whether leading zeros should be printed, and what currency symbols are used. The pattern string can be altered if a program needs to create a custom format.

## DecimalFormatSymbols Class

The class `DecimalFormatSymbols` represents the set of symbols (such as the decimal separator, the grouping separator, and so on) needed by `DecimalFormat` to format numbers. `DecimalFormat` creates for itself an instance of `DecimalFormatSymbols` from its locale data. A programmer needing to change any of these symbols can get the `DecimalFormatSymbols` object from the `DecimalFormat` object and then modify it.

## ChoiceFormat Class

The class `ChoiceFormat` is a concrete subclass of the `NumberFormat` class. The `ChoiceFormat` class allows the programmer to attach a format to a range of numbers. It is generally used in a `MessageFormat` object for handling plurals.

## MessageFormat Class

Programs often need to build messages from sequences of strings, numbers and other data. For example, the text of a message displaying the number of files on a disk drive will vary:

- The disk C contains 100 files.
- The disk D contains 1 file.
- The disk F contains 0 files.

If a message built from sequences of strings and numbers is hard-coded, it cannot be translated into other languages. For example, note the different positions of the parameters "3" and "G" in the following translations:

- The disk G contains 3 files. (English)
- Il y a 3 fichiers sur le disque G. (French)

The class `MessageFormat` provides a means to produce concatenated messages in language-neutral way. The `MessageFormat` object takes a set of objects, formats them, and then inserts the formatted strings into the pattern at the appropriate places.

## ParsePosition Class

The class `ParsePosition` is used by the `Format` class and its subclasses to keep track of the current position during parsing. The `parseObject()` method in the `Format` class requires a `ParsePosition` object as an argument.

## FieldPosition Class

The `FieldPosition` class is used by the `Format` class and its subclasses to identify fields in formatted output. One version of the `format()` method in the `Format` class requires a `FieldPosition` object as an argument.

## Locale-Sensitive String Operations

Programs frequently need to manipulate strings. Common operations on strings include searching and sorting. Some tasks, such as collating strings or finding various boundaries in text, are surprisingly difficult to get right and are even more difficult when multiple languages must be considered. The Java Platform provides classes for handling many of these common string manipulations in a locale-sensitive manner.

## Collator Class

The `Collator` class performs locale-sensitive string comparison. Programmers use this class to build searching and alphabetical sorting routines for natural language text. `Collator` is an abstract base class. Its subclasses implement specific collation strategies. One subclass, `RuleBasedCollator`, is applicable to a wide set of languages. Other subclasses may be created to handle more specialized needs.

## RuleBasedCollator Class

The `RuleBasedCollator` class, which is a concrete subclass of the `Collator` class, provides a simple, data-driven, table collator. Using `RuleBasedCollator`, a programmer can create a customized table-based collator. For example, a programmer can build a collator that will ignore (or notice) uppercase letters, accents, and Unicode combining characters.

## CollationElementIterator Class

The `CollationElementIterator` class is used as an iterator to walk through each character of an international string. Programmers use the iterator to return the ordering priority of the positioned character. The ordering priority of a character, or key, defines how a character is collated in the given `Collator` object. The `CollationElementIterator` class is used by the `compare()` method of the `RuleBasedCollator` class.

## CollationKey Class

A `CollationKey` object represents a string under the rules of a specific `Collator` object. Comparing two `CollationKey` objects returns the relative order of the strings they represent. Using `CollationKey` objects to compare strings is generally faster than using the `Collator.compare()` method. Thus, when the strings must be compared multiple times, for example when sorting a list of strings, it is more efficient to use `CollationKey` objects.

## BreakIterator Class

The `BreakIterator` class indirectly implements methods for finding the position of the following types of boundaries in a string of text:

- potential line break
- sentence
- word
- character

The conventions on where to break lines, sentences, words, and characters vary from one language to another. Since the `BreakIterator` class is locale-sensitive, it can be used by programs that perform text operations. For example, consider a word processing program that can highlight a character, cut a word, move the cursor to the next sentence, or word-wrap at a line ending. This word processing program would use break iterators to determine the logical boundaries in text, enabling it to perform text operations in a locale-sensitive manner.

## StringCharacterIterator Class

The `StringCharacterIterator` class provides the ability to iterate over a string of Unicode characters in a bidirectional manner. This class uses a cursor to move within a range of text, and can return individual characters or their index values. The `StringCharacterIterator` class implements the character iterator functionality of the `CharacterIterator` interface.

## CharacterIterator Interface

The `CharacterIterator` interface defines a protocol for bidirectional iteration over Unicode characters. Classes should implement this interface if they want to move about within a range of text and return individual Unicode characters or their index values. `CharacterIterator` is useful when performing character searches.

## Normalizer Class

The `Normalizer` class provides methods to transform Unicode text into an equivalent composed or decomposed form. The class supports the *Unicode Normalization Forms* defined by the Unicode standard.

## Locale-Sensitive Services SPIs

Locale sensitive services provided by classes in the `java.text` and `java.util` packages can be extended by implementing locale-sensitive services SPIs for locales the Java runtime has not yet supported.

The extension mechanism is not supported and SPI implementations for internationalization functions in the `java.text.spi`, `java.util.spi`, and `java.awt.im.spi` packages will be loaded from the application's classpath.

In addition to localized symbols or names for the `Currency`, `Locale`, and `TimeZone` classes in the `java.util` package, implementations of the following classes in the `java.text` package can be plugged in with the SPIs.

- `BreakIterator`

- [Collator](#)
- [DateFormat](#)
- [DateFormatSymbols](#)
- [DecimalFormatSymbols](#)
- [NumberFormat](#)

## Character Encoding Conversion

The Java platform uses Unicode as its native character encoding; however, many Java programs still need to handle text data in other encodings. Java therefore provides a set of classes that convert many standard character encodings to and from Unicode. Java programs that need to deal with non-Unicode text data convert that data into Unicode, process the data as Unicode, then convert the result back to the external character encoding. The [InputStreamReader](#) and [OutputStreamWriter](#) classes provide methods that can convert between other character encodings and Unicode.

## Supported Encodings

The [InputStreamReader](#), [OutputStreamWriter](#), and [String](#) classes can convert between Unicode and the set of character encodings listed in [Supported Encodings](#).

## Stream I/O

The Java Platform provides features in the `java.io` package to improve the handling of character data: the [Reader](#) and [Writer](#) classes, and an enhancement to the [PrintStream](#) class.

## Reader and Writer Classes

The [Reader](#) and [Writer](#) class hierarchies provide the ability to perform I/O operations on character streams. These hierarchies parallel the [InputStream](#) and [OutputStream](#) class hierarchies, but operate on streams of characters rather than streams of bytes. Character streams make it easy to write programs that are not dependent upon a specific character encoding, and are therefore easier to internationalize. The [Reader](#) and [Writer](#) classes also have the ability to convert between Unicode and other character encodings.

## PrintStream Class

The [PrintStream](#) class produces output using the system's default character encoding and line terminator. This change allows methods such as `System.out.println()` to act more reasonably with non-ASCII data.

## Charset Package

The `java.nio.charset` package provides the underpinnings for character encoding conversion. Applications can use its classes to fine-tune the behavior of built-in character converters. Developers can also create custom converters for character encodings that are not supported by built-in character converters, using the `java.nio.charset.spi` package.



## Input Methods

Input methods are software components that let the user enter text in ways other than simple typing on a keyboard. They are commonly used to enter Japanese, Chinese, or Korean - languages using thousands of different characters - on keyboards with far fewer keys. However, the Java platform also supports input methods for other languages and the use of entirely different input mechanisms, such as handwriting or speech recognition.

The Java platform enables the use of native input methods provided by the host operating system, such as Windows or Linux, as well as the implementation and use of input methods written in the Java programming language.

The term input methods does not refer to class methods of the Java programming language.

## Input Method Support in Swing

The Swing text components provide an integrated user interface for text input using input methods. Depending on the locale, one of two input styles is used. With on-the-spot (inline) input, the style used for most locales, the input methods insert the text directly into the text component while the text is being composed. With below-the-spot input, the style used for Chinese locales, a separate composition window is used, which is positioned automatically to be near the point where the text is to be inserted after being committed.

An application using Swing text components does not have to coordinate the interaction between the text components and input methods. However, it should call `InputContext.endComposition` when all text must be committed, such as when a document is saved or printed.

## Input Method Framework

The input method framework enables the collaboration between text editing components and input methods in entering text. Programmers who develop text editing components or input methods use this framework. Other application developers generally make only minimal use of it. For example, they should call `InputContext.endComposition` when all text must be committed, such as when a document is saved or printed.

# 2

## Supported Encodings

The `java.io.InputStreamReader`, `java.io.OutputStreamWriter`, `java.lang.String` classes, and classes in the `java.nio.charset` package can convert between Unicode and a number of other character encodings. The supported encodings vary between different implementations of the Java SE Platform. The class description for `java.nio.charset.Charset` lists the encodings that any implementation of the Java SE platform is required to support.

The following tables show the encoding sets supported by this version of the Oracle Java SE platform. The canonical names used by the `java.nio` APIs are in many cases not the same as those used in the `java.io` and `java.lang` APIs.

### Basic Encoding Set (contained in `java.base` module)

| Canonical Name for <code>java.nio</code> API | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases   | Description  |
|--|--|--|--|
| CESU-8                                       | CESU8  | CESU8 csCESU-8   | Unicode CESU-8   |
| GB18030                                      | GB18030  | gb18030-2022 or<br>gb18030-2000 if the<br>system property and<br>value<br><code>jdk.charset.GB18030<br/>=2000</code> are specified | Simplified Chinese, PRC<br>standard                              |
| IBM00858                                     | Cp858  | cp858 ccsid00858<br>cp00858 858 PC-<br>Multilingual-850+euro   | Variant of Cp850 with<br>Euro character                          |
| IBM437                                       | Cp437  | cp437 ibm437 ibm-437<br>437 cspc8codepage437<br>windows-437  | MS-DOS United States,<br>Australia, New Zealand,<br>South Africa |
| IBM775                                       | Cp775  | cp775 ibm775 ibm-775<br>775  | PC Baltic  |
| IBM850                                       | Cp850  | cp850 ibm-850 ibm850<br>850 cspc850multilingual  | MS-DOS Latin-1   |
| IBM852                                       | Cp852  | cp852 ibm852 ibm-852<br>852 csPCp852   | MS-DOS Latin-2   |
| IBM855                                       | Cp855  | cp855 ibm-855 ibm855<br>855 csppc855   | IBM Cyrillic   |
| IBM857                                       | Cp857  | cp857 ibm857 ibm-857<br>857 csIBM857   | IBM Turkish  |
| IBM862                                       | Cp862  | cp862 ibm862 ibm-862<br>862 csIBM862<br>cspc862latinhebrew   | PC Hebrew  |
| IBM866                                       | Cp866  | cp866 ibm866 ibm-866<br>866 csIBM866   | MS-DOS Russian   |

| Canonical Name for java.nio API | Canonical Name for java.io API and java.lang API | Alias or Aliases  | Description                                     |
|---------------------------------|--|---|---|
| ISO-8859-1                      | ISO8859_1  | iso-ir-100 ISO_8859-1 latin1 I1 IBM819 cp819 csISOLatin1 819 IBM-819 ISO8859_1 ISO_8859-1:1987 ISO_8859_1 8859_1 ISO8859-1                                  | ISO-8859-1, Latin Alphabet No. 1                |
| ISO-8859-13                     | ISO8859_13                                       | iso8859_13 8859_13 iso_8859-13 ISO8859-13   | Latin Alphabet No. 7                            |
| ISO-8859-15                     | ISO8859_15                                       | ISO_8859-15 Latin-9 csISO885915 8859_15 ISO-8859-15 ISO8859_15 ISO8859-15 IBM923 IBM-923 cp923 923 LATIN0 LATIN9 L9 csISOLatin0 csISOLatin9 ISO8859_15_FDIS | Latin Alphabet No. 9                            |
| ISO-8859-16                     | ISO8859_16                                       | iso-ir-226 ISO_8859-16:2001 ISO_8859-16 latin10 I10 csISO885916   | Latin Alphabet No. 10 or South-Eastern European |
| ISO-8859-2                      | ISO8859_2  | iso8859_2 8859_2 iso-ir-101 ISO_8859-2 ISO_8859-2:1987 ISO8859-2 latin2 I2 ibm912 ibm-912 cp912 912 csISOLatin2   | Latin Alphabet No. 2                            |
| ISO-8859-4                      | ISO8859_4  | iso8859_4 iso8859-4 8859_4 iso-ir-110 ISO_8859-4 ISO_8859-4:1988 latin4 I4 ibm914 ibm-914 cp914 914 csISOLatin4   | Latin Alphabet No. 4                            |
| ISO-8859-5                      | ISO8859_5  | iso8859_5 8859_5 iso-ir-144 ISO_8859-5 ISO_8859-5:1988 ISO8859-5 cyrillic ibm915 ibm-915 cp915 915 csISOLatinCyrillic                                       | Latin/Cyrillic Alphabet                         |
| ISO-8859-7                      | ISO8859_7  | iso8859_7 8859_7 iso-ir-126 ISO_8859-7 ISO_8859-7:1987 ELOT_928 ECMA-118 greek greek8 csISOLatinGreek sun_eu_greek ibm813 ibm-813 813 cp813 iso8859-7       | Latin/Greek Alphabet (ISO-8859-7:2003)          |

| Canonical Name for java.nio API | Canonical Name for java.io API and java.lang API | Alias or Aliases  | Description  |
|---------------------------------|--|---|--|
| ISO-8859-9                      | ISO8859_9  | iso8859_9 8859_9 iso-ir-148 ISO_8859-9 ISO_8859-9:1989 ISO8859-9 latin5 I5 ibm920 ibm-920 920 cp920 csISOLatin5                     | Latin Alphabet No. 5   |
| KOI8-R                          | KOI8_R   | koi8_r koi8 cskoi8r   | KOI8-R, Russian  |
| KOI8-U                          | KOI8_U   | koi8_u  | KOI8-U, Ukrainian  |
| US-ASCII                        | ASCII  | iso-ir-6 ANSI_X3.4-1986 ISO_646.irv:1991 ASCII ISO646-US us IBM367 cp367 csASCII default 646 iso_646.irv:1983 ANSI_X3.4-1968 ascii7 | American Standard Code for Information Interchange   |
| UTF-16                          | UTF-16   | UTF_16 utf16 unicode UnicodeBig   | Sixteen-bit Unicode (or UCS) Transformation Format, byte order identified by an optional byte-order mark |
| UTF-16BE                        | UnicodeBigUnmarked                               | UTF_16BE ISO-10646-UCS-2 X-UTF-16BE UnicodeBigUnmarked  | Sixteen-bit Unicode (or UCS) Transformation Format, big-endian byte order                                |
| UTF-16LE                        | UnicodeLittleUnmarked                            | UTF_16LE X-UTF-16LE UnicodeLittleUnmarked   | Sixteen-bit Unicode (or UCS) Transformation Format, little-endian byte order                             |
| UTF-32                          | UTF-32   | UTF_32 UTF32  | 32-bit Unicode (or UCS) Transformation Format, byte order identified by an optional byte-order mark      |
| UTF-32BE                        | UTF-32BE   | UTF_32BE X-UTF-32BE   | 32-bit Unicode (or UCS) Transformation Format, big-endian byte order                                     |
| UTF-32LE                        | UTF-32LE   | UTF_32LE X-UTF-32LE   | 32-bit Unicode (or UCS) Transformation Format, little-endian byte order                                  |
| UTF-8                           | UTF8   | UTF8 unicode-1-1-utf-8  | Eight-bit Unicode (or UCS) Transformation Format   |
| windows-1250                    | Cp1250   | cp1250 cp5346   | Windows Eastern European   |
| windows-1251                    | Cp1251   | cp1251 cp5347 ansi-1251   | Windows Cyrillic   |
| windows-1252                    | Cp1252   | cp1252 cp5348 ibm-1252 ibm1252  | Windows Latin-1  |
| windows-1253                    | Cp1253   | cp1253 cp5349   | Windows Greek  |
| windows-1254                    | Cp1254   | cp1254 cp5350   | Windows Turkish  |
| windows-1257                    | Cp1257   | cp1257 cp5353   | Windows Baltic   |
| x-IBM737                        | Cp737  | cp737 ibm737 ibm-737 737  | PC Greek   |

| Canonical Name for <code>java.nio</code> API | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases          | Description  |
|--|--|---------------------------|--|
| x-IBM874                                     | Cp874  | cp874 ibm874 ibm-874 874  | IBM Thai   |
| x-UTF-16LE-BOM                               | UnicodeLittle  | UnicodeLittle             | Sixteen-bit Unicode (or UCS) Transformation Format, little-endian byte order, with byte-order mark |
| X-UTF-32BE-BOM                               | X-UTF-32BE-BOM   | UTF_32BE_BOM UTF-32BE-BOM | 32-bit Unicode (or UCS) Transformation Format, big-endian byte order, with byte-order mark         |
| X-UTF-32LE-BOM                               | X-UTF-32LE-BOM   | UTF_32LE_BOM UTF-32LE-BOM | 32-bit Unicode (or UCS) Transformation Format, little-endian byte order, with byte-order mark      |

## Extended Encoding Set (contained in `jdk.charsets` module)

| Canonical Name for <code>java.nio</code> API | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases  | Description  |
|--|--|---|--|
| Big5<br>Big5-HKSCS                           | Big5<br>Big5_HKSCS   | csBig5<br>Big5_HKSCS big5hk<br>big5-hkscs big5hkscs   | Big5, Traditional Chinese<br>Big5 with Hong Kong extensions, Traditional Chinese (incorporating 2001 revision) |
| EUC-JP                                       | EUC_JP   | euc_jp eucjis eucjp<br>Extended_UNIX_Code_<br>Packed_Format_for_Jap<br>anese<br>csEUCPkdFmtjapanese<br>x-euc-jp x-eucjp | JISX 0201, 0208 and 0212, EUC encoding Japanese  |
| EUC-KR                                       | EUC_KR   | euc_kr ksc5601 euckr<br>ks_c_5601-1987<br>ksc5601-1987<br>ksc5601_1987 ksc_5601<br>csEUCKR 5601                         | KS C 5601, EUC encoding, Korean  |
| GB2312                                       | EUC_CN   | gb2312 gb2312-80<br>gb2312-1980 euc-cn<br>euccn x-EUC-CN<br>EUC_CN  | GB2312, EUC encoding, Simplified Chinese   |
| GBK<br>IBM01140                              | GBK<br>Cp1140  | windows-936 CP936<br>cp1140 ccsid01140<br>cp01140 1140 ebcdic-<br>us-037+euro   | GBK, Simplified Chinese<br>Variant of Cp037 with Euro character  |
| IBM01141                                     | Cp1141   | cp1141 ccsid01141<br>cp01141 1141 ebcdic-<br>de-273+euro  | Variant of Cp273 with Euro character   |

| Canonical Name for <code>java.nio</code> API | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases  | Description   |
|--|--|---|---|
| IBM01142                                     | Cp1142   | cp1142 ccsid01142<br>cp01142 1142 ebcdic-<br>no-277+euro ebcdic-<br>dk-277+euro   | Variant of Cp277 with Euro character                                      |
| IBM01143                                     | Cp1143   | cp1143 ccsid01143<br>cp01143 1143 ebcdic-<br>fi-278+euro ebcdic-<br>se-278+euro   | Variant of Cp278 with Euro character                                      |
| IBM01144                                     | Cp1144   | cp1144 ccsid01144<br>cp01144 1144 ebcdic-<br>it-280+euro  | Variant of Cp280 with Euro character                                      |
| IBM01145                                     | Cp1145   | cp1145 ccsid01145<br>cp01145 1145 ebcdic-<br>es-284+euro  | Variant of Cp284 with Euro character                                      |
| IBM01146                                     | Cp1146   | cp1146 ccsid01146<br>cp01146 1146 ebcdic-<br>gb-285+euro  | Variant of Cp285 with Euro character                                      |
| IBM01147                                     | Cp1147   | cp1147 ccsid01147<br>cp01147 1147 ebcdic-<br>fr-277+euro  | Variant of Cp297 with Euro character                                      |
| IBM01148                                     | Cp1148   | cp1148 ccsid01148<br>cp01148 1148 ebcdic-<br>international-500+euro   | Variant of Cp500 with Euro character                                      |
| IBM01149                                     | Cp1149   | cp1149 ccsid01149<br>cp01149 1149 ebcdic-<br>s-871+euro   | Variant of Cp871 with Euro character                                      |
| IBM037                                       | Cp037  | cp037 ibm037 ebcdic-<br>cp-us ebcdic-cp-ca<br>ebcdic-cp-wt ebcdic-cp-<br>nl csIBM037 cs-ebcdic-<br>cp-us cs-ebcdic-cp-ca<br>cs-ebcdic-cp-wt cs-<br>ebcdic-cp-nl ibm-037<br>ibm-37 cpibm37 037 | USA, Canada (Bilingual, French), Netherlands, Portugal, Brazil, Australia |
| IBM1026                                      | Cp1026   | cp1026 ibm1026<br>ibm-1026 1026   | IBM Latin-5, Turkey   |
| IBM1047                                      | Cp1047   | cp1047 ibm-1047 1047  | Latin-1 character set for EBCDIC hosts                                    |
| IBM273                                       | Cp273  | cp273 ibm273 ibm-273<br>273   | IBM Austria, Germany  |
| IBM277                                       | Cp277  | cp277 ibm277 ibm-277<br>277   | IBM Denmark, Norway   |
| IBM278                                       | Cp278  | cp278 ibm278 ibm-278<br>278 ebcdic-sv ebcdic-cp-<br>se csIBM278   | IBM Finland, Sweden   |
| IBM280                                       | Cp280  | cp280 ibm280 ibm-280<br>280   | IBM Italy   |
| IBM284                                       | Cp284  | cp284 ibm284 ibm-284<br>284 csIBM284 cpibm284   | IBM Catalan/Spain, Spanish Latin America                                  |

| Canonical Name for <code>java.nio</code> API | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases   | Description  |
|--|--|--|--|
| IBM285                                       | Cp285  | cp285 ibm285 ibm-285<br>285 ebcdic-cp-gb<br>ebcdic-gb csIBM285<br>cpibm285 | IBM United Kingdom, Ireland  |
| IBM290                                       | Cp290  | cp290 ibm290 ibm-290<br>csIBM290 EBCDIC-JP-<br>kana 290                    | IBM Japanese Katakana Host Extended SBCS   |
| IBM297                                       | Cp297  | cp297 ibm297 ibm-297<br>297 ebcdic-cp-fr<br>cpibm297 csIBM297              | IBM France   |
| IBM420                                       | Cp420  | cp420 ibm420 ibm-420<br>ebcdic-cp-ar1 420<br>csIBM420                      | IBM Arabic   |
| IBM424                                       | Cp424  | cp424 ibm424 ibm-424<br>424 ebcdic-cp-he<br>csIBM424                       | IBM Hebrew   |
| IBM500                                       | Cp500  | cp500 ibm500 ibm-500<br>500 ebcdic-cp-ch ebcdic-<br>cp-bh csIBM500         | EBCDIC 500V1   |
| IBM860                                       | Cp860  | cp860 ibm860 ibm-860<br>860 csIBM860                                       | MS-DOS Portuguese  |
| IBM861                                       | Cp861  | cp861 ibm861 ibm-861<br>861 csIBM861 cp-is                                 | MS-DOS Icelandic   |
| IBM863                                       | Cp863  | cp863 ibm863 ibm-863<br>863 csIBM863                                       | MS-DOS Canadian French   |
| IBM864                                       | Cp864  | cp864 ibm864 ibm-864<br>864 csIBM864                                       | PC Arabic  |
| IBM865                                       | Cp865  | cp865 ibm865 ibm-865<br>865 csIBM865                                       | MS-DOS Nordic  |
| IBM868                                       | Cp868  | cp868 ibm868 ibm-868<br>868 cp-ar csIBM868                                 | MS-DOS Pakistan  |
| IBM869                                       | Cp869  | cp869 ibm869 ibm-869<br>869 cp-gr csIBM869                                 | IBM Modern Greek   |
| IBM870                                       | Cp870  | cp870 ibm870 ibm-870<br>870 ebcdic-cp-roece<br>ebcdic-cp-yu csIBM870       | IBM Multilingual Latin-2   |
| IBM871                                       | Cp871  | cp871 ibm871 ibm-871<br>871 ebcdic-cp-is<br>csIBM871                       | IBM Iceland  |
| IBM918                                       | Cp918  | cp918 ibm-918 918<br>ebcdic-cp-ar2   | IBM Pakistan (Urdu)  |
| IBM-Thai                                     | Cp838  | cp838 ibm838 ibm-838<br>838  | IBM Thailand extended SBCS   |
| ISO-2022-CN                                  | ISO2022CN  | ISO2022CN<br>csISO2022CN   | GB2312 and CNS11643 in ISO 2022 CN form, Simplified and Traditional Chinese (conversion to Unicode only) |
| ISO-2022-JP                                  | ISO2022JP  | iso2022jp jis<br>csISO2022JP<br>jis_encoding<br>csjisencoding              | JIS X 0201, 0208, in ISO 2022 form, Japanese   |

| Canonical Name for <code>java.nio</code> API | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases  | Description  |
|--|--|---|--|
| ISO-2022-JP-2                                | ISO2022JP2   | <code>csISO2022JP2</code><br><code>iso2022jp2</code>  | JIS X 0201, 0208, 0212 in ISO 2022 form, Japanese  |
| ISO-2022-KR                                  | ISO2022KR  | ISO2022KR<br><code>csISO2022KR</code>   | ISO 2022 KR, Korean  |
| ISO-8859-3                                   | ISO8859_3  | <code>iso8859_3</code> <code>8859_3</code><br><code>ISO_8859-3:1988</code> <code>iso-ir-109</code> <code>ISO_8859-3</code><br><code>ISO8859-3</code> <code>latin3</code> <code>I3</code><br><code>ibm913</code> <code>ibm-913</code> <code>cp913</code><br><code>913</code> <code>csISOLatin3</code>  | Latin Alphabet No. 3   |
| ISO-8859-6                                   | ISO8859_6  | <code>iso8859_6</code> <code>8859_6</code> <code>iso-ir-127</code> <code>ISO_8859-6</code><br><code>ISO_8859-6:1987</code><br><code>ISO8859-6</code> <code>ECMA-114</code><br><code>ASMO-708</code> <code>arabic</code><br><code>ibm1089</code> <code>ibm-1089</code><br><code>cp1089</code> <code>1089</code><br><code>csISOLatinArabic</code> | Latin/Arabic Alphabet  |
| ISO-8859-8                                   | ISO8859_8  | <code>iso8859_8</code> <code>8859_8</code> <code>iso-ir-138</code> <code>ISO_8859-8</code><br><code>ISO_8859-8:1988</code><br><code>ISO8859-8</code> <code>cp916</code> <code>916</code><br><code>ibm916</code> <code>ibm-916</code> <code>hebrew</code><br><code>csISOLatinHebrew</code>   | Latin/Hebrew Alphabet  |
| JIS_X0201                                    | JIS_X0201  | JIS0201 JIS_X0201<br>X0201<br><code>csHalfWidthKatakana</code>  | JIS X 0201   |
| JIS_X0212-1990                               | JIS0212  | JIS0212 <code>jis_x0212-1990</code><br>x0212 <code>iso-ir-159</code><br><code>csISO159JISX02121990</code>   | JIS X 0212   |
| Shift_JIS                                    | SJIS   | <code>sjis</code> <code>shift_jis</code> <code>shift-jis</code><br><code>ms_kanji</code> <code>x-sjis</code><br><code>csShiftJIS</code>   | Shift-JIS, Japanese  |
| TIS-620                                      | TIS620   | <code>tis620</code> <code>tis620.2533</code>  | TIS620, Thai   |
| <code>windows-1255</code>                    | <code>Cp1255</code>  | <code>cp1255</code>   | Windows Hebrew   |
| <code>windows-1256</code>                    | <code>Cp1256</code>  | <code>cp1256</code>   | Windows Arabic   |
| <code>windows-1258</code>                    | <code>Cp1258</code>  | <code>cp1258</code>   | Windows Vietnamese   |
| <code>windows-31j</code>                     | MS932  | MS932 <code>windows-932</code><br><code>csWindows31J</code>   | Windows Japanese   |
| <code>x-Big5-HKSCS-2001</code>               | <code>x-Big5-HKSCS-2001</code>   | <code>Big5_HKSCS_2001</code><br><code>big5hk-2001</code> <code>big5-hkscs-2001</code> <code>big5-hkscs:unicode3.0</code><br><code>big5hkscs-2001</code>   | Big5 with Hong Kong Supplementary Character Set, 2001 revision   |
| <code>x-Big5-Solaris</code>                  | <code>Big5_Solaris</code>  | <code>Big5_Solaris</code>   | Big5 with seven additional Hanzi ideograph character mappings for the Solaris <code>zh_TW.BIG5</code> locale |
| <code>x-euc-jp-linux</code>                  | EUC_JP_LINUX   | <code>euc_jp_linux</code> <code>euc-jp-linux</code>   | JIS X 0201, 0208, EUC encoding Japanese  |



| Canonical Name for <code>java.nio</code> API | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases   | Description   |
|--|--|--|---|
| <code>x-eucJP-Open</code>                    | <code>EUC_JP_Solaris</code>  | <code>EUC_JP_Solaris</code> <code>eucJP-open</code>  | JISX 0201, 0208, 0212, EUC encoding Japanese                              |
| <code>x-EUC-TW</code>                        | <code>EUC_TW</code>  | <code>euc_tw</code> <code>euctw</code> <code>cns11643</code> <code>EUC-TW</code>   | CNS11643 (Plane 1-7,15), EUC encoding, Traditional Chinese                |
| <code>x-IBM1006</code>                       | <code>Cp1006</code>  | <code>cp1006</code> <code>ibm1006</code> <code>ibm-1006</code> <code>1006</code>   | IBM AIX Pakistan (Urdu)   |
| <code>x-IBM1025</code>                       | <code>Cp1025</code>  | <code>cp1025</code> <code>ibm1025</code> <code>ibm-1025</code> <code>1025</code>   | IBM Multilingual Cyrillic: Bulgaria, Bosnia, Herzegovina, Macedonia (FYR) |
| <code>x-IBM1046</code>                       | <code>Cp1046</code>  | <code>cp1046</code> <code>ibm1046</code> <code>ibm-1046</code> <code>1046</code>   | IBM Arabic - Windows  |
| <code>x-IBM1097</code>                       | <code>Cp1097</code>  | <code>cp1097</code> <code>ibm1097</code> <code>ibm-1097</code> <code>1097</code>   | IBM Iran (Farsi)/Persian  |
| <code>x-IBM1098</code>                       | <code>Cp1098</code>  | <code>cp1098</code> <code>ibm1098</code> <code>ibm-1098</code> <code>1098</code>   | IBM Iran (Farsi)/Persian (PC)   |
| <code>x-IBM1112</code>                       | <code>Cp1112</code>  | <code>cp1112</code> <code>ibm1112</code> <code>ibm-1112</code> <code>1112</code>   | IBM Latvia, Lithuania   |
| <code>x-IBM1122</code>                       | <code>Cp1122</code>  | <code>cp1122</code> <code>ibm1122</code> <code>ibm-1122</code> <code>1122</code>   | IBM Estonia   |
| <code>x-IBM1123</code>                       | <code>Cp1123</code>  | <code>cp1123</code> <code>ibm1123</code> <code>ibm-1123</code> <code>1123</code>   | IBM Ukraine   |
| <code>x-IBM1124</code>                       | <code>Cp1124</code>  | <code>cp1124</code> <code>ibm1124</code> <code>ibm-1124</code> <code>1124</code>   | IBM AIX Ukraine   |
| <code>x-IBM1129</code>                       | <code>Cp1129</code>  | <code>cp1129</code> <code>ibm1129</code> <code>ibm-1129</code> <code>1129</code>   | IBM AIX Vietnamese  |
| <code>x-IBM1166</code>                       | <code>Cp1166</code>  | <code>cp1166</code> <code>ibm1166</code> <code>ibm-1166</code> <code>1166</code>   | IBM Cyrillic Multilingual with euro for Kazakhstan                        |
| <code>x-IBM1364</code>                       | <code>Cp1364</code>  | <code>cp1364</code> <code>ibm1364</code> <code>ibm-1364</code> <code>1364</code>   | IBM EBCDIC KS X 1005-1  |
| <code>x-IBM1381</code>                       | <code>Cp1381</code>  | <code>cp1381</code> <code>ibm1381</code> <code>ibm-1381</code> <code>1381</code>   | IBM OS/2, DOS People's Republic of China (PRC)                            |
| <code>x-IBM1383</code>                       | <code>Cp1383</code>  | <code>cp1383</code> <code>ibm1383</code> <code>ibm-1383</code> <code>1383</code> <code>ibmeuccn</code> <code>ibm-euccn</code> <code>cpeuccn</code> | IBM AIX People's Republic of China (PRC)                                  |
| <code>x-IBM300</code>                        | <code>Cp300</code>   | <code>cp300</code> <code>ibm300</code> <code>ibm-300</code> <code>300</code>   | IBM Japanese Latin Host Double-Byte                                       |
| <code>x-IBM33722</code>                      | <code>Cp33722</code>   | <code>cp33722</code> <code>ibm33722</code> <code>ibm-33722</code> <code>ibm-5050</code> <code>ibm-33722_vascii_vpua</code> <code>33722</code>      | IBM-eucJP - Japanese (superset of 5050)                                   |
| <code>x-IBM833</code>                        | <code>Cp833</code>   | <code>cp833</code> <code>ibm833</code> <code>ibm-833</code>  | IBM Korean Host Extended SBCS   |
| <code>x-IBM834</code>                        | <code>Cp834</code>   | <code>cp834</code> <code>ibm834</code> <code>834</code> <code>ibm-834</code>   | IBM EBCDIC DBCS-only Korean   |
| <code>x-IBM856</code>                        | <code>Cp856</code>   | <code>cp856</code> <code>ibm-856</code> <code>ibm856</code> <code>856</code>   | IBM Hebrew  |
| <code>x-IBM875</code>                        | <code>Cp875</code>   | <code>cp875</code> <code>ibm875</code> <code>ibm-875</code> <code>875</code>   | IBM Greek   |

| Canonical Name for <code>java.nio</code> API | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases   | Description  |
|--|--|--|--|
| x-IBM921                                     | Cp921  | cp921 ibm921 ibm-921 921                                       | IBM Latvia, Lithuania (AIX, DOS)   |
| x-IBM922                                     | Cp922  | cp922 ibm922 ibm-922 922                                       | IBM Estonia (AIX, DOS)   |
| x-IBM930                                     | Cp930  | cp930 ibm930 ibm-930 930                                       | Japanese Katakana-Kanji mixed with 4370 UDC, superset of 5026                    |
| x-IBM933                                     | Cp933  | cp933 ibm933 ibm-933 933                                       | Korean Mixed with 1880 UDC, superset of 5029                                     |
| x-IBM935                                     | Cp935  | cp935 ibm935 ibm-935 935                                       | Simplified Chinese Host mixed with 1880 UDC, superset of 5031                    |
| x-IBM937                                     | Cp937  | cp937 ibm937 ibm-937 937                                       | Traditional Chinese Host mixed with 6204 UDC, superset of 5033                   |
| x-IBM939                                     | Cp939  | cp939 ibm939 ibm-939 939                                       | Japanese Latin Kanji mixed with 4370 UDC, superset of 5035                       |
| x-IBM942                                     | Cp942  | cp942 ibm942 ibm-942 942                                       | IBM OS/2 Japanese, superset of Cp932   |
| x-IBM942C                                    | Cp942C   | cp942C ibm942C ibm-942C 942C cp932 ibm932 ibm-932 932 x-ibm932 | Variant of Cp942   |
| x-IBM943                                     | Cp943  | cp943 ibm943 ibm-943 943                                       | IBM OS/2 Japanese, superset of Cp932 and Shift-JIS                               |
| x-IBM943C                                    | Cp943C   | cp943C ibm943C ibm-943C 943C                                   | Variant of Cp943   |
| x-IBM948                                     | Cp948  | cp948 ibm948 ibm-948 948                                       | OS/2 Chinese (Taiwan) superset of 938  |
| x-IBM949                                     | Cp949  | cp949 ibm949 ibm-949 949                                       | PC Korean  |
| x-IBM949C                                    | Cp949C   | cp949C ibm949C ibm-949C 949C                                   | Variant of Cp949   |
| x-IBM950                                     | Cp950  | cp950 ibm950 ibm-950 950                                       | PC Chinese (Hong Kong, Taiwan)   |
| x-IBM964                                     | Cp964  | cp964 ibm964 ibm-964 ibm-euctw 964                             | AIX Chinese (Taiwan)   |
| x-IBM970                                     | Cp970  | cp970 ibm970 ibm-970 ibm-euckR 970                             | AIX Korean   |
| x-ISCI191                                    | ISCI191  | iscii ST_SEV_358-88 iso-ir-153 csISO153GOST1976874 ISCI191     | ISCI191 encoding of Indic scripts  |
| x-ISO-2022-CN-CNS                            | ISO2022CN_CNS  | ISO2022CN_CNS ISO-2022-CN-CNS                                  | CNS11643 in ISO 2022 CN form, Traditional Chinese (conversion from Unicode only) |

| Canonical Name for <code>java.nio</code> API         | Canonical Name for <code>java.io</code> API and <code>java.lang</code> API | Alias or Aliases   | Description   |
|--|--|--|---|
| <code>x-ISO-2022-CN-GB</code>                        | <code>ISO2022CN_GB</code>  | <code>ISO2022CN_GB</code><br><code>ISO-2022-CN-GB</code>   | GB2312 in ISO 2022 CN form, Simplified Chinese (conversion from Unicode only)         |
| <code>x-iso-8859-11</code><br><code>x-JIS0208</code> | <code>x-iso-8859-11</code><br><code>JIS0208</code>                         | <code>iso-8859-11</code> <code>iso8859_11</code><br><code>JIS0208</code><br><code>JIS_C6226-1983</code> <code>iso-ir-87</code> <code>x0208</code><br><code>JIS_X0208-1983</code><br><code>csISO87JISX0208</code> | Latin/Thai Alphabet<br>JIS X 0208   |
| <code>x-JISAutoDetect</code>                         | <code>JISAutoDetect</code>   | <code>JISAutoDetect</code>   | Detects and converts from Shift-JIS, EUC-JP, ISO 2022 JP (conversion to Unicode only) |
| <code>x-Johab</code>                                 | <code>x-Johab</code>   | <code>ksc5601-1992</code><br><code>ksc5601_1992</code> <code>ms1361</code><br><code>johab</code>   | Korean, Johab character set   |
| <code>x-MacArabic</code>                             | <code>MacArabic</code>   | <code>MacArabic</code>   | Macintosh Arabic  |
| <code>x-MacCentralEurope</code>                      | <code>MacCentralEurope</code>  | <code>MacCentralEurope</code>  | Macintosh Latin-2   |
| <code>x-MacCroatian</code>                           | <code>MacCroatian</code>   | <code>MacCroatian</code>   | Macintosh Croatian  |
| <code>x-MacCyrillic</code>                           | <code>MacCyrillic</code>   | <code>MacCyrillic</code>   | Macintosh Cyrillic  |
| <code>x-MacDingbat</code>                            | <code>MacDingbat</code>  | <code>MacDingbat</code>  | Macintosh Dingbat   |
| <code>x-MacGreek</code>                              | <code>MacGreek</code>  | <code>MacGreek</code>  | Macintosh Greek   |
| <code>x-MacHebrew</code>                             | <code>MacHebrew</code>   | <code>MacHebrew</code>   | Macintosh Hebrew  |
| <code>x-MacIceland</code>                            | <code>MacIceland</code>  | <code>MacIceland</code>  | Macintosh Iceland   |
| <code>x-MacRoman</code>                              | <code>MacRoman</code>  | <code>MacRoman</code>  | Macintosh Roman   |
| <code>x-MacRomania</code>                            | <code>MacRomania</code>  | <code>MacRomania</code>  | Macintosh Romania   |
| <code>x-MacSymbol</code>                             | <code>MacSymbol</code>   | <code>MacSymbol</code>   | Macintosh Symbol  |
| <code>x-MacThai</code>                               | <code>MacThai</code>   | <code>MacThai</code>   | Macintosh Thai  |
| <code>x-MacTurkish</code>                            | <code>MacTurkish</code>  | <code>MacTurkish</code>  | Macintosh Turkish   |
| <code>x-MacUkraine</code>                            | <code>MacUkraine</code>  | <code>MacUkraine</code>  | Macintosh Ukraine   |
| <code>x-MS932_0213</code>                            | <code>x-MS950-HKSCS</code>   | <code>MS932-0213</code><br><code>MS932_0213</code><br><code>MS932:2004</code><br><code>windows-932-0213</code><br><code>windows-932:2004</code>  | <code>Shift_JISX0213</code><br>Windows MS932 Variant                                  |
| <code>x-MS950-HKSCS</code>                           | <code>MS950_HKSCS</code>   | <code>MS950_HKSCS</code>   | Windows Traditional Chinese with Hong Kong extensions                                 |
| <code>x-MS950-HKSCS-XP</code>                        | <code>x-mswin-936</code>   | <code>MS950_HKSCS_XP</code>  | HKSCS Windows XP Variant  |
| <code>x-mswin-936</code>                             | <code>MS936</code>   | <code>ms936</code> <code>ms_936</code>   | Windows Simplified Chinese  |
| <code>x-PCK</code>                                   | <code>PCK</code>   | <code>pck</code>   | Solaris version of <code>Shift_JIS</code>   |
| <code>x-SJIS_0213</code>                             | <code>x-SJIS_0213</code>   | <code>sjis-0213</code> <code>sjis_0213</code><br><code>sjis:2004</code> <code>sjis_0213:2004</code><br><code>shift_jis_0213:2004</code><br><code>shift_jis:2004</code>   | <code>Shift_JISX0213</code>   |

| Canonical Name for java.nio API | Canonical Name for java.io API and java.lang API | Alias or Aliases                    | Description                                   |
|---------------------------------|--|-------------------------------------|---|
| x-windows-50220                 | MS50220  | ms50220 cp50220                     | Windows Codepage 50220 (7-bit implementation) |
| x-windows-50221                 | MS50221  | ms50221 cp50221                     | Windows Codepage 50221 (7-bit implementation) |
| x-windows-874                   | MS874  | ms874 ms-874 windows-874            | Windows Thai                                  |
| x-windows-949                   | MS949  | ms949 windows949 windows-949 ms_949 | Windows Korean                                |
| x-windows-950                   | MS950  | ms950 windows-950                   | Windows Traditional Chinese                   |
| x-windows-iso2022jp             | windows-iso2022jp                                | windows-iso2022jp                   | Variant ISO-2022-JP (MS932 based)             |

## Printing Charset Information

The following application prints the aliases of each charset supported by Java SE:

```
import java.nio.charset.*;

class DisplayCharsetAliases {
    public static void main(String[] args) {
        System.out.println("Charset -> Aliases");
        System.out.println("=====");
        for (Charset cs : Charset.availableCharsets().values()) {
            System.out.println(cs.name() + " -> " + cs.aliases());
        }
    }
}
```

## Default Charset

The default charset is UTF-8. However, in JDK 17 and earlier releases, the default charset depends on the host and the user.

Standard Java APIs use the default charset unless you specify one. These APIs include:

- In the package `java.io`, the classes `InputStreamReader`, `FileReader`, `OutputStreamWriter`, `FileWriter`, and `PrintStream`, which define constructors to create readers, writers, and print streams that encode or decode using the default charset
- In the package `java.util`, the classes `Formatter` and `Scanner`, which define constructors whose results use the default charset

 **Note:**

The standard output stream `System.out` and the standard error output stream `System.err` don't use the default charset; they use the charset specified by `Console.charset()`.

Specify the encoding for `System.out` and `System.err` with the system properties `stdout.encoding` and `stderr.encoding`, respectively. The default values of these system properties depend on the platform. The default values take on the value of the `native.encoding` property when the platform does not provide streams for the console.

## Default Charset for JDK 17 and Earlier Releases

In JDK 17 and earlier releases, the default charset is determined when the Java runtime starts. On macOS, the default charset is UTF-8 except in the POSIX C locale. On other operating systems, it depends on the user's locale and the default encoding. For example, on Windows, it's a codepage-based charset such as `windows-1252` or `windows-31j`. The method `java.nio.charsets.Charset.defaultCharset()` returns the default charset.

You can run the following command to determine the default charset of your JDK:

```
java -XshowSettings:properties -version 2>&1 | grep file.encoding
```

## Changing the JDK's Default Charset

You can set the value of the `file.encoding` system property on the command line to one of the following values to specify that the JDK's default charset is UTF-8 or the default charset is determined as in JDK 17 and earlier releases:

- `UTF-8`: The default charset is UTF-8.
- `COMPAT`: The default charset is determined as in JDK 17 and earlier releases.

Other values for `file.encoding` are not supported.

 **Note:**

Before deploying your application on a JDK whose default charset is UTF-8, check if it has any charset issues by running it on a JDK whose default charset is *not* UTF-8 with the following command:

```
java -Dfile.encoding=UTF-8 <your application>
```

## Running Java Applications on JDK Whose Default Charset Is Determined by Environment

JDK 17 introduced the system property `native.encoding`. Use this property to obtain the underlying host environment's character encoding name, especially if you specified that your JDK determines the default charset as in JDK 17 and earlier releases.



### Note:

Setting the value of the system property `native.encoding` through the command line or with the method `System.setProperty()` has no effect.

The following example obtains the default charset from the system property `native.encoding`. Note that you can run this example on any JDK release; if the system property `native.encoding` hasn't been defined, then the example obtains the default charset from the method `Charset.defaultCharset()`:

```
String encoding = System.getProperty("native.encoding");
Charset cs = (encoding != null) ? Charset.forName(encoding) :
Charset.defaultCharset();
```

If your application expects the default charset to be determined as in JDK 17 and earlier releases, then use this obtained charset as a constructor argument for objects that rely on a charset, for example:

```
var reader = new FileReader("file.txt", cs);
```



### Note:

The method call `Charset.forName("default")` throws an `UnsupportedCharsetException`. Use `Charset.forName("US-ASCII")` or `Charset.defaultCharset()` instead. (In JDK 17 and earlier releases, `Charset.forName("default")` produces the same result as `Charset.forName("US-ASCII")`.)

The value of `native.encoding` affects the value of `file.encoding`:

- If `file.encoding` is set to `COMPAT` on the command line, then the run-time value of `file.encoding` will be the same as the run-time value of `native.encoding`.
- If `file.encoding` is set to `UTF-8` on the command line, then the run-time value of `file.encoding` may differ from the run-time value of `native.encoding`.

## Ensuring Source File Encoding Is Compatible with Your JDK

The `javac` compiler assumes that `.java` source files are encoded with the default charset unless configured otherwise with the `-encoding` option.

Consequently, before compiling an application on a JDK whose default charset is UTF-8, check for charset issues by compiling your application with the following command:

```
javac -encoding UTF-8 <source files of your application>
```

Alternatively, if you prefer to save your source files with an encoding other than UTF-8, specify in the `-encoding` option the value of the `native.encoding` system property.

# 3

## Supported Calendars

The core of the Date-Time API is the `java.time` package. The classes defined in `java.time` base their calendar system on the ISO calendar, which is the world standard for representing date and time. The ISO calendar follows the proleptic Gregorian rules. There are also non-ISO calendars predefined in `java.time.chrono` package: the Japanese, Hijrah, Minguo, and Thai Buddhist calendars. For more about the Date-Time API, see the [Internationalization Trail](#) in the Java Tutorials.



# 4

## Supported Fonts

Different OS platforms may provide fonts that are implemented using different font technologies. To support cross-platform use, the Java SE API defines five families of "logical" fonts that can safely be used by an application using any Java SE implementation. The names of these families are defined in the [Font](#) class description.

Additionally a Java SE implementation may expose the platform fonts to be used directly by name. These fonts are called "physical" fonts.

For more information on the terminology used here, see the [Font](#) class description.

- [Support for Physical Fonts](#)
- [Support for Logical Fonts](#)

### Support for Physical Fonts

The JDK supports TrueType, OpenType, and PostScript Type 1 fonts.

Physical fonts need to be installed in locations known to the Java runtime environment. The JDK locates fonts in the standard font locations defined by the host operating system.

You can add physical fonts that use a supported font technology by installing them in a way supported by the host operating system. The recommended location to add per-user fonts on Linux is the `$HOME/.fonts` directory which is searched by the platform's `libfontconfig`, and which is in turn used by the JDK.

### Support for Logical Fonts

Typically one logical font maps to several physical fonts in order to cover a larger range of code points than is possible with a single font. Logical fonts are mapped to physical fonts in implementation-dependent ways, and can vary from platform to platform and from release to release.

Font configuration files are used by some implementations to handle the mapping, see [Font Configuration Files](#):

- Current releases for Windows always use font configuration files.
- The macOS implementation always ignores font configuration files.
- Releases for Linux use font configuration files only if there is an exact match for the OS version, otherwise font configuration files are ignored and platform APIs are used to populate the logical fonts.

# 5

## Font Configuration Files

The Java platform defines five logical font names that every implementation must support: `Serif`, `SansSerif`, `Monospaced`, `Dialog`, and `DialogInput`. These logical font names are mapped to physical fonts in implementation dependent ways.

One way the Oracle JDK maps logical font names to physical fonts is by using *font configuration* files. There may be several files to support different mappings depending on the host operating system version. The files are distributed with the JDK installation. You can edit or create your own font configuration files to adjust the mappings to your particular system setup, however these must be placed in `conf/fonts`, and are subject to implementation notes discussed below.

Font configuration files come in two formats: a properties format and a binary format. The properties format is described in detail in this document and can be used for user-defined configurations. The binary format is undocumented and used only for the JDK's predefined configurations; the corresponding files in properties format are available for reference as files with the `.properties.src` extension.

## Supported Platforms

Font configuration files are implementation dependent. Not all implementations of the Java Platform use them, and the format and content vary between runtime environments and releases. The macOS implementation does not use font configuration files, as the mapping is hard-coded in the source and cannot be changed.

The Oracle JDK supports font configuration files on the host operating system as follows:

- For Windows, font configuration files are required.
- For macOS, font configuration files are unsupported.
- For Linux, the Oracle JDK is moving away from providing custom font configuration files on Linux platforms, as they are difficult to keep up to date across distributions and versions. A distribution that has control over the fonts on the system can continue to provide this custom file. If the Java runtime finds a custom file that exactly matches the distribution and version, that file is used. If no exact match is found, the file is dynamically created at runtime. These generated files are placed in a location determined by the implementation. They should be considered implementation internal: they are not user editable and do not follow the syntax as described in this specification.

## Loading Font Configuration Files

The JDK places any files that it provides in `$JDKHOME/lib`. Do not modify that location. Instead, put any updates or custom versions of the configuration files in `$JDKHOME/conf/fonts`.

On platforms that support font configuration files, the runtime will look first in `$JDKHOME/conf/fonts`. In other words, a user-supplied file is preferred if it is a match.

The font configuration file for a host operating system is located as follows:

- *JavaHome* - the JDK directory, as given by the `java.home` system property.
- *OS* - a string identifying an operating system variant:
  - For Windows, empty.
  - For Linux, "RedHat", "SuSE", etc.
- *Version* - a string identifying the operating system version.

The runtime uses the first of the following files it finds:

```
JavaHome/lib/fontconfig.OS.Version.properties
JavaHome/lib/fontconfig.OS.Version.bfc
JavaHome/lib/fontconfig.OS.properties
JavaHome/lib/fontconfig.OS.bfc
JavaHome/lib/fontconfig.Version.properties
JavaHome/lib/fontconfig.Version.bfc
JavaHome/lib/fontconfig.properties
JavaHome/lib/fontconfig.bfc
```

Files with a `.properties` suffix are assumed to be properties files as specified by the [Properties](#) class and are loaded through that class. Files without this suffix are assumed to be in binary format.

## Names Used in Font Configuration Files

Throughout the font configuration files, a number of different names are used:

- *LogicalFontName* - one of the five logical font names: `serif`, `sansserif`, `monospaced`, `dialog`, and `dialoginput`. In font configuration files, these names are always in lowercase.
- *StyleName* - one of the four standard font styles: `plain`, `bold`, `italic`, and `bolditalic`. Again, these names are always in lowercase.
- *PlatformFontName* - the name of a physical font, in a format typically used on the platform:
  - On Windows, a font face name, such as "Courier New" or "\uad74\ub9bc".
  - On Linux, an xld name, such as "-monotype-times new roman-regular-r---\*-%d-\*-\*-p-\*-iso8859-1". Note that "%d" is used for the font size - the actual font size is filled in at runtime.
- *CharacterSubsetName* - a name for a subset of the Unicode character set which certain component fonts can render.
  - For Windows, the following names are predefined: `alphabetic`, `arabic`, `chinese-ms936`, `chinese-gb18030`, `chinese-ms950`, `chinese-hkscs`, `cyrillic-iso8859-5`, `cyrillic-cp1251`, `cyrillic-koi8-r`, `devanagari`, `dingbats`, `greek`, `hebrew`, `japanese`, `korean`, `latin`, `symbol`, `thai`.
  - For Linux, the following names are predefined: `arabic`, `chinese-gb2312`, `chinese-gbk`, `chinese-gb18030-0`, `chinese-gb18030-1`, `chinese-cns11643-1`, `chinese-cns11643-2`, `chinese-cns11643-3`, `chinese-big5`, `chinese-hkscs`, `cyrillic`, `devanagari`, `dingbats`, `greek`, `hebrew`, `japanese-x0201`, `japanese-x0208`, `japanese-x0212`, `korean`, `korean-johab`, `latin-1`, `latin-2`, `latin-4`, `latin-5`, `latin-7`, `latin-9`, `symbol`, `thai`.

A font configuration file may define additional names to identify additional character subsets.

- *Encoding* - the canonical name of the default encoding, as provided by `java.nio.charset.Charset.defaultCharset().name()`.
- *Language* - the language of the initial default locale.
- *Country* - the country of the initial default locale.

## Properties for All Platforms

Properties that are applicable to all platforms enable you to specify the font configuration format version, component font mappings, search sequences, exclusion ranges, proportional fonts, font file names, and appended font path.

### Version Property

The version property identifies the font configuration format version. This document specifies version 1.

The complete property has the form:

```
version=1
```

### Component Font Mappings

Component font mapping properties describe which physical font to use to render characters from a given character subset with a given logical font in a given style.

The keys have the forms:

```
allfonts.CharacterSubsetName  
LogicalFontName.StyleName.CharacterSubsetName
```

The first form is used if the same font is used for a character subset independent of logical font and style (in this case, the font rendering engines apply algorithmic styles to the font). The second form is used if different physical fonts are used for a character subset for different logical fonts and styles. In this case, properties must be specified for each combination of logical font and style, so 20 properties for one character subset. If a property of the first form is present for a character subset, then properties of the second form for the same character subset are ignored.

The values are platform font names, as described in [Names Used in Font Configuration Files](#).

Since the character subsets supported by given fonts often overlap, separate search sequence properties are used to define in which order to try the fonts when rendering a character.

### Search Sequences

The Java runtime uses sequence properties to determine search sequences for the five logical fonts. However, a font configuration file may specify properties that are specific to a combination of encoding, language, and country, and the runtime will then use a lookup to determine the search sequence property for each logical font.

The keys have the form:

```
sequence.allfonts.Encoding.Language.Country  
sequence.LogicalFontName.Encoding.Language.Country  
sequence.allfonts.Encoding.Language  
sequence.LogicalFontName.Encoding.Language  
sequence.allfonts.Encoding  
sequence.LogicalFontName.Encoding  
sequence.allfonts  
sequence.LogicalFontName
```

The `allfonts` forms are used if the sequence is used for all five logical fonts. The forms specifying logical font names are used if different sequences are used for different logical fonts.

For each logical font, the Java runtime uses the property value with the first of the above keys. This property determines the primary search sequence for the logical font.

The file may also define a single fallback search sequence. The key for the fallback search sequence property is:

```
sequence.fallback
```

The values of all search sequence properties have the form:

```
SearchSequenceValue:  
  CharacterSubsetName  
  CharacterSubsetName , SearchSequenceValue
```

The primary search sequence properties specify character subset names for required fonts, which are used for both AWT and 2D font rendering. The fallback search sequence property gives character subset names for optional fonts, which are used as fallbacks for all logical fonts, but only for 2D font rendering. On Windows, if there is a system EUDC (End User Defined Characters) font registered with Windows, the runtime automatically adds this font as well as a fallback font for 2D rendering.

The sequence properties determine in which sequence component fonts are tried to render a given character. For example, given the following properties:

```
sequence.monospaced=japanese,alphabetic  
sequence.fallback=korean  
monospaced.plain.alphabetic=Arial  
monospaced.plain.japanese=MSGothic  
monospaced.plain.korean=Gulim
```

The runtime will first attempt to render a character with the MSGothic font. If that font doesn't provide a glyph for the character, it will attempt the Arial font. For 2D rendering, it will also try the Gulim font as well as any TrueType, OpenType, or Type 1 fonts in the system's standard font locations. For 2D rendering on Windows, if there is a system EUDC font registered with Windows, the runtime will also try this EUDC font.

When calculating font metrics for a logical font without reference to a string, only the required fonts are taken into consideration. For the example above, the `FontMetrics.getMaxDescent` method would return results based on the MSGothic and Arial fonts, but not the Gulim font. In this way, simple user interface elements such as buttons, which sometimes calculate their size based on font metrics, are not affected by an extended

list of component fonts which their labels usually don't use. On the other hand, text components typically calculate metrics based on the text they contain and thus will obtain correct results.

The sequence properties that the runtime obtains for the five logical fonts should list the same character subsets, but may list them in different order.

## Exclusion Ranges

The exclusion range properties specify Unicode character ranges which should be excluded from being rendered with the fonts corresponding to a given character subset. This is used if a font with a large character repertoire needs to be placed early in the search sequence (for example, for performance reasons), but some characters that it supports should be drawn with a different font instead. These properties are optional, so there's at most one per character subset.

The keys have the form:

```
exclusion.CharacterSubsetName
```

The values have the form:

```
ExclusionRangeValue:
```

```
Range
```

```
Range , ExclusionRangeValue
```

```
Range:
```

```
Char - Char
```

```
Char:
```

```
HexDigit HexDigit HexDigit HexDigit
```

```
HexDigit HexDigit HexDigit HexDigit HexDigit HexDigit
```

A *Char* is a Unicode character represented as a hexadecimal value.

## Proportional Fonts

The proportional font properties describe the relationship between proportional and non-proportional variants of otherwise equivalent fonts. These properties are used to implement preferences specified by the [GraphicsEnvironment.preferProportionalFonts](#) method.

The keys have the form:

```
proportional.PlatformFontName
```

Space characters in the platform font name must be replaced with underscore characters (`_`).

The values have the form:

```
PlatformFontName
```

In values, space characters are left unmodified.

Each property indicates that the font named in the value is the proportional equivalent of the font named in the key, and also that the font named in the key is the non-proportional equivalent of the font named in the value.

## Font File Names

Font file name properties provide the names of the files containing the physical fonts used in the font configuration file. File names are required for all physical fonts on Windows and recommended for all physical fonts on Linux.

The keys have the form:

```
filename.PlatformFontName
```

Space characters in the platform font name must be replaced with underscore characters (`_`).

The values are the file names of the files containing the fonts. On Windows, simple file names are used; and the runtime environment looks for each file first in its own `lib/fonts` directory if one exists, then in the Windows fonts directory. On Linux, absolute path names, path names starting with `"$JRE_LIB_FONTS"` for the runtime environment's own `lib/fonts` directory, or `xlfd` names are used.



### Note:

Oracle JDK does not deliver any fonts in the `lib/fonts` location.

## Appended Font Path

The Java runtime can automatically determine a number of directories that contain font files, such as its own `lib/fonts` directory if one exists, or the Windows fonts folder. Additional directories can be specified to be appended to the font path.

The key has the form:

```
appendedfontpath
```

The value has the form:

```
AppendedFontPathValue:  
  Directory  
  Directory PathSeparator AppendedFontPathValue
```

The path separator is the platform dependent value of `java.io.File.pathSeparator`.

## Properties for Windows

There are no platform-specific properties for Windows. However, there is a special form of the character subset name used in search sequences. The name "alphabetic" can take a suffix indicating the character encoding associated with the subset:

```
alphabetic  
alphabetic/default  
alphabetic/1252
```

This information is only used for AWT, not for 2D. The `/default` suffix restricts use of the component fonts for this character subset to the character set of the default encoding; the `/1252` suffix to the Windows-1252 character set. For accessing component font mappings and exclusion ranges, the character encoding suffix is omitted. For all other character subsets, the AWT character encoding is determined internally by the Java runtime.

## Property for Linux

The only property that is specific to Linux is the AWT font path, which identifies platform directories that should be added to the X11 server font path.

The keys have the form:

```
awtfontpath.CharacterSubsetName
```

The values have the form:

```
AWTFontPathValue:  
  Directory  
  Directory : AWTFontPathValue
```

The directories must be valid X11 font directories. The Java runtime ensures that the directories for all character subsets of a primary search sequence found by the search sequence lookup (see [Search Sequences](#)) are part of the X11 font path. The implementation assumes that all logical fonts use the same set of character subsets for a given environment of encoding, language, and country.