Data.Char

Copyright	(c) The University of Glasgow 2001
License	BSD-style (see the file libraries/base/LICENSE)
Maintainer	libraries@haskell.org
Stability	stable
Portability	portable
Safe Haskell	Trustworthy
Language	Haskell2010

The Char type and associated operations.

Documentation

`data Char :: *` | #

The character type `Char` is an enumeration whose values represent Unicode (or equivalently ISO/IEC 10646) characters (see http://www.unicode.org/ for details). This set extends the ISO 8859-1 (Latin-1) character set (the first 256 characters), which is itself an extension of the ASCII character set (the first 128 characters). A character literal in Haskell has type `Char`.

To convert a `Char` to or from the corresponding `Int` value defined by Unicode, use `toEnum` and `fromEnum` from the `Enum` class respectively (or equivalently `ord` and `chr`).

Instances

`Bounded Char`	# Source	
`Enum Char`	# Source	
`Eq Char`		
`Data Char`	# Source	
`Ord Char`		
`Read Char`	# Source	
`Show Char`	# Source	
`Ix Char`	# Source	
`Storable Char`	# Source	
`IsChar Char`	# Source	
`PrintfArg Char`	# Source	
`Functor (URec Char)`	# Source	
`Foldable (URec Char)`	# Source	
`Traversable (URec Char)`	# Source	
`Generic1 (URec Char)`	# Source	
`Eq (URec Char p)`	#	
`Ord (URec Char p)`	#	
`Show (URec Char p)`	# Source	
`Generic (URec Char p)`	# Source	
`data URec Char`	# Source	Used for marking occurrences of `Char#`
`type Rep1 (URec Char)`	# Source	
`type Rep (URec Char p)`	# Source	

Character classification

Unicode characters are divided into letters, numbers, marks, punctuation, symbols, separators (including spaces) and others (including control characters).

`isControl :: Char -> Bool` # Source

Selects control characters, which are the non-printing characters of the Latin-1 subset of Unicode.

`isSpace :: Char -> Bool` # Source

Returns `True` for any Unicode space character, and the control characters `\t`, `\n`, `\r`, `\f`, `\v`.

`isLower :: Char -> Bool` # Source

Selects lower-case alphabetic Unicode characters (letters).

`isUpper :: Char -> Bool` # Source

Selects upper-case or title-case alphabetic Unicode characters (letters). Title case is used by a small number of letter ligatures like the single-character form of *Lj*.

`isAlpha :: Char -> Bool` # Source

Selects alphabetic Unicode characters (lower-case, upper-case and title-case letters, plus letters of caseless scripts and modifiers letters). This function is equivalent to `isLetter`.

`isAlphaNum :: Char -> Bool` # Source

Selects alphabetic or numeric digit Unicode characters.

Note that numeric digits outside the ASCII range are selected by this function but not by `isDigit`. Such digits may be part of identifiers but are not used by the printer and reader to represent numbers.

`isPrint :: Char -> Bool` # Source

Selects printable Unicode characters (letters, numbers, marks, punctuation, symbols and spaces).

`isDigit :: Char -> Bool` # Source

Selects ASCII digits, i.e. `'0'..'9'`.

`isOctDigit :: Char -> Bool` # Source

Selects ASCII octal digits, i.e. `'0'..'7'`.

`isHexDigit :: Char -> Bool` # Source

Selects ASCII hexadecimal digits, i.e. `'0'..'9'`, `'a'..'f'`, `'A'..'F'`.

`isLetter :: Char -> Bool` # Source

Selects alphabetic Unicode characters (lower-case, upper-case and title-case letters, plus letters of caseless scripts and modifiers letters). This function is equivalent to `isAlpha`.

This function returns `True` if its argument has one of the following `GeneralCategory`s, or `False` otherwise:

- `UppercaseLetter`
- `LowercaseLetter`
- `TitlecaseLetter`
- `ModifierLetter`
- `OtherLetter`

These classes are defined in the Unicode Character Database, part of the Unicode standard. The same document defines what is and is not a "Letter".

Examples

`isMark :: Char -> Bool` # Source

Selects Unicode mark characters, for example accents and the like, which combine with preceding characters.

This function returns `True` if its argument has one of the following `GeneralCategory`s, or `False` otherwise:

- `NonSpacingMark`
- `SpacingCombiningMark`
- `EnclosingMark`

These classes are defined in the Unicode Character Database, part of the Unicode standard. The same document defines what is and is not a "Mark".

Examples

`isNumber :: Char -> Bool` # Source

Selects Unicode numeric characters, including digits from various scripts, Roman numerals, et cetera.

This function returns `True` if its argument has one of the following `GeneralCategory`s, or `False` otherwise:

- `DecimalNumber`
- `LetterNumber`
- `OtherNumber`

These classes are defined in the Unicode Character Database, part of the Unicode standard. The same document defines what is and is not a "Number".

Examples

`isPunctuation :: Char -> Bool` # Source

Selects Unicode punctuation characters, including various kinds of connectors, brackets and quotes.

This function returns `True` if its argument has one of the following `GeneralCategory`s, or `False` otherwise:

- `ConnectorPunctuation`
- `DashPunctuation`
- `OpenPunctuation`
- `ClosePunctuation`
- `InitialQuote`
- `FinalQuote`
- `OtherPunctuation`

These classes are defined in the Unicode Character Database, part of the Unicode standard. The same document defines what is and is not a "Punctuation".

Examples

`isSymbol :: Char -> Bool` # Source

Selects Unicode symbol characters, including mathematical and currency symbols.

This function returns `True` if its argument has one of the following `GeneralCategory`s, or `False` otherwise:

- `MathSymbol`
- `CurrencySymbol`
- `ModifierSymbol`
- `OtherSymbol`

These classes are defined in the Unicode Character Database, part of the Unicode standard. The same document defines what is and is not a "Symbol".

Examples

`isSeparator :: Char -> Bool` # Source

Selects Unicode space and separator characters.

This function returns `True` if its argument has one of the following `GeneralCategory`s, or `False` otherwise:

- `Space`
- `LineSeparator`
- `ParagraphSeparator`

These classes are defined in the Unicode Character Database, part of the Unicode standard. The same document defines what is and is not a "Separator".

Examples

Subranges

`isAscii :: Char -> Bool` # Source

Selects the first 128 characters of the Unicode character set, corresponding to the ASCII character set.

`isLatin1 :: Char -> Bool` # Source

Selects the first 256 characters of the Unicode character set, corresponding to the ISO 8859-1 (Latin-1) character set.

`isAsciiUpper :: Char -> Bool` # Source

Selects ASCII upper-case letters, i.e. characters satisfying both `isAscii` and `isUpper`.

`isAsciiLower :: Char -> Bool` # Source

Selects ASCII lower-case letters, i.e. characters satisfying both `isAscii` and `isLower`.

Unicode general categories

`data GeneralCategory` # Source

Unicode General Categories (column 2 of the UnicodeData table) in the order they are listed in the Unicode standard (the Unicode Character Database, in particular).

Examples

Constructors

`UppercaseLetter`	Lu: Letter, Uppercase
`LowercaseLetter`	Ll: Letter, Lowercase

`TitlecaseLetter`	Lt: Letter, Titlecase
`ModifierLetter`	Lm: Letter, Modifier
`OtherLetter`	Lo: Letter, Other
`NonSpacingMark`	Mn: Mark, Non-Spacing
`SpacingCombiningMark`	Mc: Mark, Spacing Combining
`EnclosingMark`	Me: Mark, Enclosing
`DecimalNumber`	Nd: Number, Decimal
`LetterNumber`	Nl: Number, Letter
`OtherNumber`	No: Number, Other
`ConnectorPunctuation`	Pc: Punctuation, Connector
`DashPunctuation`	Pd: Punctuation, Dash
`OpenPunctuation`	Ps: Punctuation, Open
`ClosePunctuation`	Pe: Punctuation, Close
`InitialQuote`	Pi: Punctuation, Initial quote
`FinalQuote`	Pf: Punctuation, Final quote
`OtherPunctuation`	Po: Punctuation, Other
`MathSymbol`	Sm: Symbol, Math
`CurrencySymbol`	Sc: Symbol, Currency
`ModifierSymbol`	Sk: Symbol, Modifier
`OtherSymbol`	So: Symbol, Other
`Space`	Zs: Separator, Space
`LineSeparator`	Zl: Separator, Line
`ParagraphSeparator`	Zp: Separator, Paragraph
`Control`	Cc: Other, Control
`Format`	Cf: Other, Format
`Surrogate`	Cs: Other, Surrogate
`PrivateUse`	Co: Other, Private Use
`NotAssigned`	Cn: Other, Not Assigned

Instances

`Bounded GeneralCategory`	# Source
`Enum GeneralCategory`	# Source
`Eq GeneralCategory`	# Source
`Ord GeneralCategory`	# Source
`Read GeneralCategory`	# Source
`Show GeneralCategory`	# Source
`Ix GeneralCategory`	# Source

`generalCategory` `:: Char -> GeneralCategory` # Source

The Unicode general category of the character. This relies on the `Enum` instance of `GeneralCategory`, which must remain in the same order as the categories are presented in the Unicode standard.

Examples

Case conversion

toUpper :: Char -> Char # Source

Convert a letter to the corresponding upper-case letter, if any. Any other character is returned unchanged.

toLower :: Char -> Char # Source

Convert a letter to the corresponding lower-case letter, if any. Any other character is returned unchanged.

toTitle :: Char -> Char # Source

Convert a letter to the corresponding title-case or upper-case letter, if any. (Title case differs from upper case only for a small number of ligature letters.) Any other character is returned unchanged.

Single digit characters

digitToInt :: Char -> Int # Source

Convert a single digit `Char` to the corresponding `Int`. This function fails unless its argument satisfies `isHexDigit`, but recognises both upper- and lower-case hexadecimal digits (that is, `'0'`..`'9'`, `'a'`..`'f'`, `'A'`..`'F'`).

Examples

intToDigit :: Int -> Char # Source

Convert an `Int` in the range `0`..`15` to the corresponding single digit `Char`. This function fails on other inputs, and generates lower-case hexadecimal digits.

Numeric representations

ord :: Char -> Int # Source

The `fromEnum` method restricted to the type `Char`.

chr :: Int -> Char # Source

The `toEnum` method restricted to the type `Char`.

String representations

showLitChar :: Char -> ShowS # Source

Convert a character to a string using only printable characters, using Haskell source-language escape conventions. For example:

```
showLitChar '\n' s  =  "\\n" ++ s
```

lexLitChar :: ReadS String # Source

Read a string representation of a character, using Haskell source-language escape conventions. For example:

```
lexLitChar  "\\nHello"  =  [("\\n", "Hello")]
```

readLitChar :: ReadS Char # Source

Read a string representation of a character, using Haskell source-language escape conventions, and convert it to the character that it encodes. For example:

```
readLitChar "\\nHello"  =  [('\n', "Hello")]
```