?? | Language & Region Identifiers

? Learning Objectives

- Understand locale identifiers and their structure
- Master BCP 47 language tags and IETF standards
- Distinguish between language, region, and script
- Implement proper locale detection and fallback
- Handle special cases and edge scenarios

? What is a Locale?

A **locale** is a set of parameters that defines the user's language, region, and cultural preferences. It determines how your application formats numbers, dates, currency, and displays text.

Locale Components

Language

The primary language being used (e.g., English, Spanish, Japanese)

`en, es, ja, zh`

Region/Territory

The country or region affecting formats and conventions

`US, GB, CN, BR`

Script (Optional)

The writing system used for the language

`Latn, Cyrl, Arab, Hans`

Variant (Rare)

Specific dialectal or orthographic variations

`valencia, pinyin`

? BCP 47 Language Tags

BCP 47 (Best Current Practice 47) is the IETF standard for language tags. It defines how to construct identifiers that specify language, region, script, and variants.

BCP 47 Tag Structure

language-Script-REGION-variant

Example: `zh-Hans-CN` = Chinese (Simplified script) as used in China

Component	Format	Standard	Example
Language	2-3 lowercase letters	ISO 639	`en, es, zh, ar`
Script	4 letters, title case	ISO 15924	`Latn, Cyrl, Arab, Hans`
Region	2 uppercase letters or 3 digits	ISO 3166-1	`US, GB, CN, 001`
Variant	5-8 alphanumeric	IANA registry	`valencia, posix`

Common BCP 47 Examples

BCP 47 Tag	Description	Use Case
`en-US`	English (United States)	MM/DD/YYYY, $ before amount
`en-GB`	English (United Kingdom)	DD/MM/YYYY, £ before amount
`es-ES`	Spanish (Spain)	Uses € and European Spanish
`es-MX`	Spanish (Mexico)	Uses $ and Mexican Spanish
`zh-Hans-CN`	Chinese (Simplified, China)	Simplified characters, Mainland
`zh-Hant-TW`	Chinese (Traditional, Taiwan)	Traditional characters, Taiwan
`ar-SA`	Arabic (Saudi Arabia)	RTL, Arabic numerals, Saudi riyal

`pt-BR`	Portuguese (Brazil)	Brazilian Portuguese, R$ currency
`pt-PT`	Portuguese (Portugal)	European Portuguese, € currency
`fr-CA`	French (Canada)	Canadian French, $ currency

? Why Both Language AND Region Matter

Same language, different formats:

- `en-US` vs `en-GB`: "color" vs "colour", $ vs £, MM/DD vs DD/MM
- `es-ES` vs `es-MX`: € vs $, "ordenador" vs "computadora"
- `fr-FR` vs `fr-CA`: € vs $, some vocabulary differences
- `pt-BR` vs `pt-PT`: Significant spelling and vocabulary differences

? Locale Detection & Fallback

How do you determine a user's locale? There are multiple strategies, and you should use them in order of priority.

Locale Detection Priority

1. **User Preference (Explicit Setting):** Highest priority — user has explicitly selected their locale in settings
2. **URL Parameter:** `?lang=en-GB` or `/en-gb/page` — useful for switching without login
3. **Cookie/Session:** Previously saved preference from this device
4. **Browser Accept-Language Header:** `Accept-Language: en-US,en;q=0.9,es;q=0.8`
5. **IP Geolocation:** Infer from user's location (least reliable, privacy concerns)
6. **Default Fallback:** Your application's default locale (usually `en-US`)

Locale Detection Code Examples

JavaScript (Browser)

```
// Get browser's locale preferences
const userLocales = navigator.languages || [navigator.language];
console.log(userLocales);
// ? ["en-US", "en", "es"]

// Get primary locale
const primaryLocale = navigator.language;
console.log(primaryLocale);
// ? "en-US"
```

```
// Detect and use best available locale
function getBestLocale(supportedLocales, userPreferences) {
  // Try exact match first
  for (const userLocale of userPreferences) {
    if (supportedLocales.includes(userLocale)) {
      return userLocale;
    }
  }

  // Try language-only match (en-GB ? en-US)
  for (const userLocale of userPreferences) {
    const language = userLocale.split('-')[0];
    const match = supportedLocales.find(l => l.startsWith(language));
    if (match) return match;
  }

  // Fallback to default
  return supportedLocales[0];
}

const supported = ['en-US', 'es-ES', 'fr-FR', 'de-DE'];
const userPrefs = ['en-GB', 'en', 'es'];
const bestLocale = getBestLocale(supported, userPrefs);
console.log(bestLocale);  // ? "en-US" (language match)
```

Python (Server-side)

```
from flask import request
from babel import Locale, negotiate_locale

# Supported locales in your application
SUPPORTED_LOCALES = ['en_US', 'es_ES', 'fr_FR', 'de_DE', 'ja_JP']
DEFAULT_LOCALE = 'en_US'

def get_user_locale():
    # 1. Check user's explicit preference (from database/session)
    user_pref = session.get('locale')
    if user_pref and user_pref in SUPPORTED_LOCALES:
        return user_pref

    # 2. Check URL parameter
    url_locale = request.args.get('lang')
    if url_locale and url_locale in SUPPORTED_LOCALES:
        return url_locale

    # 3. Negotiate from Accept-Language header
    header_locales = request.accept_languages
    best_match = negotiate_locale(
        [str(l) for l in header_locales],
        SUPPORTED_LOCALES,
        sep='_'
    )
    if best_match:
        return best_match

    # 4. Fallback to default
    return DEFAULT_LOCALE
```

```
# Usage
locale = get_user_locale()
print(f"Using locale: {locale}")
```

?? Locale Fallback Chain

Always implement a **fallback chain**. If you don't have `zh-Hant-HK` (Traditional Chinese, Hong Kong), try falling back to:

1. `zh-Hant-HK` (exact match) → not available
2. `zh-Hant` (language + script) → try this
3. `zh` (language only) → then this
4. `en` (default language) → final fallback

? Working with Locales in Code

JavaScript - Parsing and Validating Locales

```
// Check if locale is valid
function isValidLocale(locale) {
  try {
    Intl.NumberFormat(locale);
    return true;
  } catch (e) {
    return false;
  }
}

console.log(isValidLocale('en-US'));     // ? true
console.log(isValidLocale('invalid'));   // ? false

// Get canonical locale
const canonical = Intl.getCanonicalLocales('EN-us')[0];
console.log(canonical);  // ? "en-US" (normalized)

// Parse locale components
function parseLocale(tag) {
  const parts = tag.split('-');
  return {
    language: parts[0]?.toLowerCase(),
    script: parts[1]?.length === 4 ? parts[1] : undefined,
    region: parts.find(p => p.length === 2)?.toUpperCase(),
  };
}

console.log(parseLocale('zh-Hans-CN'));
// ? { language: 'zh', script: 'Hans', region: 'CN' }

// Using Intl.Locale (modern browsers)
const locale = new Intl.Locale('zh-Hans-CN');
console.log(locale.language);   // ? "zh"
console.log(locale.script);     // ? "Hans"
console.log(locale.region);     // ? "CN"
```

```
console.log(locale.baseName);   // ? "zh-Hans-CN"
```

Python - Working with Babel Locales

```
from babel import Locale, UnknownLocaleError

# Parse locale
try:
    locale = Locale.parse('zh_Hans_CN', sep='_')
    print(f"Language: {locale.language}")      # ? zh
    print(f"Script: {locale.script}")          # ? Hans
    print(f"Territory: {locale.territory}")    # ? CN
    print(f"Display name: {locale.display_name}")  # ? Chinese (Simplified, China)
except UnknownLocaleError:
    print("Invalid locale")

# Get English name for locale
locale = Locale.parse('fr_CA')
print(locale.get_display_name('en'))  # ? "French (Canada)"
print(locale.get_display_name('fr'))  # ? "français (Canada)"

# List all available locales
from babel.localedata import list as list_locales
all_locales = list_locales()
print(f"Available locales: {len(all_locales)}")
# ? Available locales: 700+

# Locale negotiation
from babel import negotiate_locale

supported = ['en_US', 'es_ES', 'fr_FR']
user_prefs = ['de_DE', 'en_GB', 'en']
best = negotiate_locale(user_prefs, supported, sep='_')
print(best)  # ? "en_US" (language fallback from en)
```

? Special Cases & Edge Scenarios

? Script Matters for Some Languages

Chinese has two writing systems:

- `zh-Hans` — Simplified Chinese (Mainland China, Singapore)
- `zh-Hant` — Traditional Chinese (Taiwan, Hong Kong, Macau)

Using just `zh` is ambiguous and can lead to displaying the wrong script!

? Language Without Region

Sometimes you have only `en` without a region. What should you do?

- **Option 1:** Use a sensible default (e.g., `en` → `en-US`)
- **Option 2:** Use language-only formatting (may not be culturally appropriate)

- **Option 3:** Detect region from IP/browser and complete the locale

? Format Separators: Underscore vs Hyphen

Different systems use different separators:

- **BCP 47 / IETF / JavaScript:** `en-US` (hyphen)
- **POSIX / Python / Java:** `en_US` (underscore)

Be prepared to convert between formats: `en-US` ↔ `en_US`

?? Don't Use Locale for Authorization

Never assume that `locale = "de-DE"` means the user is in Germany or should see Germany-specific content. Users can set any locale regardless of location. Use separate mechanisms for:

- **Locale:** Formatting preferences (how to display data)
- **Location/Region:** What content/features to show (geo-restrictions, pricing)

? Best Practices Checklist

Practice	Priority
☐ Use BCP 47 format for language tags (en-US, not en_US in APIs)	**CRITICAL**
☐ Always include both language AND region (en-US, not just en)	**HIGH**
☐ Implement locale fallback chain (zh-Hant-HK → zh-Hant → zh → en)	**CRITICAL**
☐ Let users explicitly choose their locale (don't just auto-detect)	**HIGH**
☐ Validate locale codes before using them	**HIGH**
☐ Store user's locale preference in profile/session	**MEDIUM**
☐ Use script subtag for Chinese (zh-Hans vs zh-Hant)	**CRITICAL**
☐ Don't confuse locale with user location/authorization	**CRITICAL**
☐ Test locale detection with various browser/header configurations	**HIGH**

? Additional Resources

- **BCP 47:** RFC 5646 - Tags for Identifying Languages
- **IANA Language Subtag Registry:** Official registry of language codes
- **ISO 639:** Language codes standard
- **ISO 3166-1:** Country/region codes standard
- **ISO 15924:** Script codes standard
- **Unicode CLDR:** Common Locale Data Repository
- **Intl.Locale (JavaScript):** MDN documentation
- **Babel (Python):** Locale handling library

Next Topic: Start Day of the Week →

Revision #2
Created 5 November 2025 22:59:37 by itsLittleKevin
Updated 6 November 2025 19:34:47