

# Dominando a Localização em CAP

Um Mergulho Profundo no Módulo `cds.i18n` para Node.js



O módulo `cds.i18n` é a base para internacionalização em CAP. Na maioria das vezes, o framework o utiliza automaticamente. Mas por baixo da superfície, há um poderoso conjunto de ferramentas para controle total. Vamos mergulhar.

# O Básico: Como o CAP Lida com i18n Automaticamente

O CAP fornece dois *bundles* de i18n padrão, prontos para uso, que cobrem os cenários mais comuns de localização.



## `cds.i18n.labels`

Utilizado para gerar UIs localizadas, traduzindo placeholders {i18n>...} em anotações Fiori para elementos como títulos e cabeçalhos.

### Arquivos de Origem

Carrega traduções de arquivos com o nome base i18n (ex: \_i18n/i18n\_de.properties).



## `cds.i18n.messages`

Utilizado para todas as mensagens de erro ou notificação criadas via req.reject, `req.error`, etc. Isso inclui tanto erros do framework (como validação) quanto erros personalizados.

### Arquivos de Origem

Carrega traduções de arquivos com o nome base messages (ex: \_i18n/messages\_de.properties).

# Exemplo Prático 1: Localizando UIs Fiori com `Labels`

## app/fiori-annotations.cds

```
annotate CatalogService.Books with  
@title: '{i18n>Book}';
```

O CAP automaticamente  
busca a chave 'Book' no  
bundle  
'cds.i18n.labels'.

## \_i18n/i18n.properties

```
Book = Book
```

## \_i18n/i18n\_de.properties

```
Book = Buch
```

O arquivo de  
tradução  
correspondente  
ao *locale* do  
usuário é usado  
para resolver o  
valor.

# Exemplo Prático 2: Mensagens de Erro com `messages`

O bundle `cds.i18n.messages` é usado automaticamente por `req.reject()` para fornecer mensagens de erro localizadas.

## srv/cat-service.js

```
srv.before('submitOrder', async req => {
  let { book:id, quantity } = req.data
  let {stock} = await SELECT `stock`.from
    (Books,id)
  if (stock < quantity)
    req.reject(409, 'ORDER_EXCEEDS_STOCK', { stock,
      quantity })
})
```

A chave da mensagem é usada em vez de texto fixo.

## \_i18n/messages.properties

```
ORDER_EXCEEDS_STOCK = The order of {quantity}
books exceeds available stock {stock}
```

Os parâmetros ({stock}, {quantity}) são substituídos automaticamente pelos valores passados no objeto.

# Acessando Traduções Diretamente no Código

Além do uso automático, você pode invocar os bundles padrão diretamente para obter textos localizados em sua lógica customizada. O método central é `.at(key, locale?)`.

## Exemplo 1: Usando cds repl para Consultas Rápidas

```
// Locale explícito
> cds.i18n.labels.at('CreatedAt', 'de')
//> 'Erstellt am'

// Usa o locale do contexto (cds.context.locale)
> cds.i18n.labels.at('CreatedAt')
//> 'Created At'
```

## Exemplo 2: Usando com Mensagens Parametrizadas

```
> cds.i18n.messages.at('ASSERT_FORMAT', [11,12])
//> 'Enter a value matching the pattern {1}.'
```

`at()` e `for()` são sinônimos.

# E se eu precisar de traduções além de UIs e Erros?

Sua aplicação pode precisar de textos para emails, logs, relatórios ou outros domínios de negócio que não se encaixam em `labels` ou `messages`. Solução: Crie seus próprios *bundles* de tradução com a função fábrica `cds.i18n.bundle4()`.

## Exemplo de Criação:

```
const b = cds.i18n.bundle4('yours');
```

## Uso:

```
// O uso é idêntico aos bundles padrão  
b.at('some key');
```

## Estrutura de Arquivos Correspondente

Para que o *bundle* `yours` funcione, você deve fornecer os arquivos de tradução.



\_i18n/



yours.properties



yours\_de.properties



yours\_fr.properties

# A Fábrica de Bundles: `cds.i18n.bundle4()`

`'bundle4()'` é o método recomendado para criar ou obter instâncias de `'I18nBundle'`.

## Assinatura 1: Por Nome de Arquivo (string)

```
function cds.i18n.bundle4  
(file : string, options?)
```

Cria um novo *bundle* associado ao *basename* dos arquivos `.*properties*` (ex: 'foo' para `foo.*properties*`).

## Comportamento Chave: Cache

Quando a variante de string é usada, o *bundle* criado é armazenado em cache. Chamadas subsequentes com o mesmo nome retornarão a instância em cache.

```
const b1 = cds.i18n.bundle4('foo');
//> cria um novo I18nBundle para
    'foo'
const b2 = cds.i18n.bundle4('foo');
//> retorna o anteriormente criado
b1 === b2;                                //> true
```

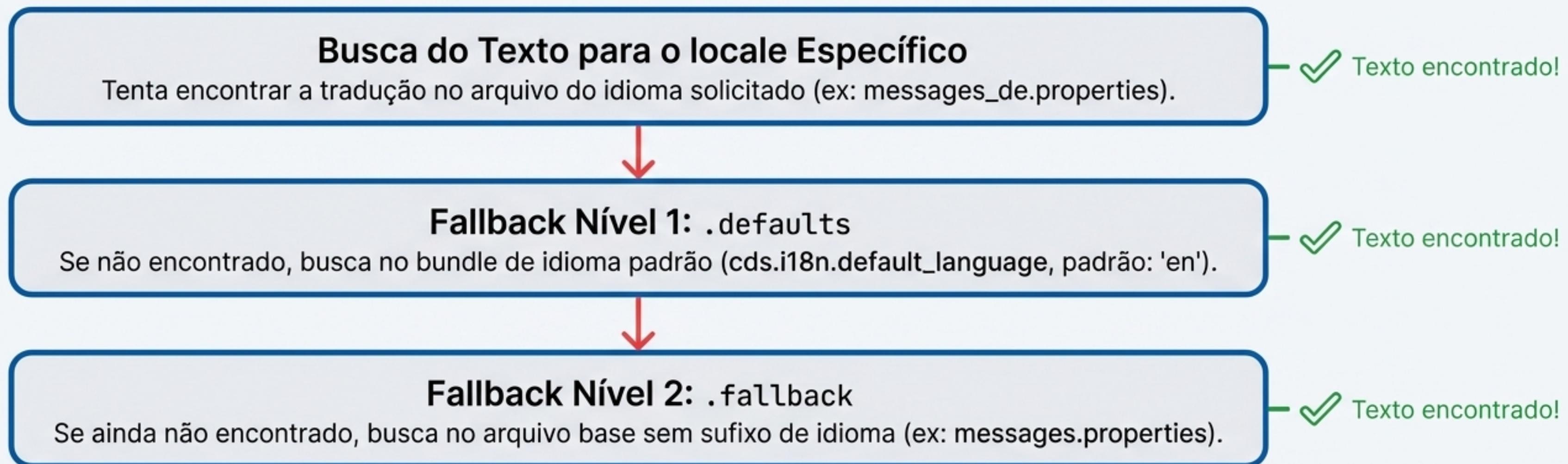
## Assinatura 2: Por Modelo CDS (CSN)

```
function cds.i18n.bundle4  
(model : CSN, options?)
```

Cria um *bundle* associado a um modelo CDS específico, útil em cenários mais complexos de componentização.

# Anatomia de um I18nBundle

Uma instância de I18nBundle fornece acesso a textos traduzidos. Sua robustez vem de um sistema de fallback em múltiplos níveis.



## Propriedade Chave: .files

Uma instância de `I18nFiles` com as pastas e arquivos encontrados para carregar o conteúdo i18n. Este é o motor por trás da descoberta de arquivos, que veremos a seguir.

# O Método Central para Obter Textos: `at()` / `for()`

Métodos principais para acessar textos traduzidos em um I18nBundle.

```
function at (key: number | string | object, locale?: string, args?: object | any[]) => string
```

## Cenário 1: Uso Básico

Se `locale` for omitido, o valor de `cds.context.locale` é usado.

```
cds.i18n.messages.at(404)      //> 'Not Found' (se locale for  
cds.i18n.messages.at(404,'de') //> 'Nicht Gefunden'
```

## Cenário 2: Usando Templates e Argumentos

Placeholders `{}` no texto são substituídos pelos valores em `args`.

```
OUT_OF_RANGE = {val} is not in range {min}..{max}
```

```
const msg = cds.i18n.messages  
msg.for('OUT_OF_RANGE', {val:0, min:1, max:11})  
//> '0 is not in range 1..11'
```

## Cenário 2: Usando Templates e Argumentos

Placeholders `{}` no texto são substituídos pelos valores em `args`.

```
OUT_OF_RANGE = {val} is not in range {min}..{max}
```



## Cenário 3: Buscando Labels para Definições CSN

Em vez de uma chave `string`, você pode passar uma definição de entidade ou elemento do modelo CDS. O CAP extrai a chave i18n das anotações `@title`, `@Common.Label`, etc.).

```
let {Books} = CatalogService.entities;  
cds.i18n.labels.at(Books); //> 'Livre' (se locale for  
'fr' e houver anotação)
```

# Os Bastidores: Como o CAP Encontra Seus Arquivos?

**Como um `I18nBundle` sabe em quais diretórios procurar por `i18n\_de.properties` ou `messages\_fr.properties`?**



## A Resposta: A classe `I18nFiles`

Instâncias desta classe são usadas através de `I18nBundle.files` para buscar e construir um dicionário de pastas e arquivos i18n que correspondem a uma dada configuração.

## O Mecanismo Padrão

Por padrão, o `I18nFiles` busca pastas na "vizinhança" dos arquivos-fonte do seu modelo (`cds.model.$sources`).

## Por que essa abordagem?

“Para encontrar conteúdo i18n de pacotes reutilizáveis com configuração zero: como pacotes reutilizáveis frequentemente vêm com seus próprios modelos CDS, nós simplesmente usamos as localizações dessas fontes .cds como pontos de partida para procurar por pastas i18n.”

# A Lógica de Busca Padrão: Na Vizinhança do Modelo

## 1. Ponto de Partida

Começa com a lista de todos os arquivos-fonte do modelo: cds.model.\$sources.

## 2. Diretórios-Fonte

Extrai os diretórios únicos onde esses arquivos estão localizados.

## 3. Busca Recursiva

Para cada diretório-fonte (em ordem reversa), o CAP:

- Procura por **subdiretórios** que correspondam à configuração cds.i18n.folders (padrão: ['\_i18n', 'i18n']).
- Se não encontrar, sobe um nível no diretório pai e repete a busca.
- Continua subindo até atingir a raiz do projeto.

## 4. Resultado

Uma lista de todas as pastas i18n encontradas, que serão usadas para carregar as traduções. A ordem reversa garante que fontes mais específicas (ex: app) sobreponham as mais genéricas (ex: db).

## Visual Principal

	_i18n	i18n
/cap/samples/node_modules/@sap/cds		
/cap/samples/common		
/cap/samples/reviews		
/cap/samples/orders		
/cap/samples/bookstore		
/cap/samples		

# Inspecionando o Resultado da Busca

Você pode ver exatamente quais pastas e arquivos um bundle encontrou usando o `cds repl`.

## Passo 1: Carregue seu modelo

```
[dev] cds repl  
> cds.model = await cds.load('bookstore')
```

## Passo 2: Inspecione a propriedade `files`

```
> cds.i18n.labels.files
```

## Saída Esperada (Exemplo)

```
{  
  '/cap/samples/node_modules/@sap/cds/_i18n':  
    ['i18n_de.properties', ...]  
  ],  
  '/cap/samples/orders/_i18n': [  
    'i18n_de.properties', ...]  
  ],  
  '/cap/samples/reviews/_i18n': [  
    'i18n_de.properties', ...]  
  ],  
  '/cap/samples/bookstore/_i18n': [  
    'i18n_de.properties', ...]  
}
```

Isso mostra a lista final de diretórios dos quais as traduções para o bundle `files` serão carregadas.

# Assumindo o Controle: Configurando a Busca de Arquivos

Você pode customizar o comportamento de busca e outras opções de i18n em seu `package.json`.

## Opções de Configuração Chave

- **cds.i18n.file**  
O nome base dos arquivos para o bundle `cds.i18n.labels`. (Padrão: 'i18n')
- **cds.i18n.default\_language**  
O locale usado como primeiro nível de fallback. (Padrão: 'en')
- **cds.i18n.folders**  
Array de nomes de pastas a serem procuradas. (Padrão: ['\_i18n', 'i18n'])

## Customizando `folders`

- **Caminhos Relativos** (ex: "\_i18n")  
São procurados na vizinhança do modelo, como visto anteriormente.
- **Caminhos Estáticos** (iniciando com /, ex: "/app/browse/webapp/i18n")  
São adicionados diretamente à lista de busca, sem o algoritmo de subida na árvore de diretórios. Útil para UIs Fiori Elements, por exemplo.

## Exemplo de Configuração `package.json`

```
"cds": {  
  "i18n": {  
    "folders": [  
      "_i18n",  
      "/app/browse/webapp/i18n"  
    ],  
    "default_language": "fr"  
  }  
}
```



Alterar essas configurações afeta TODOS os *bundles*, incluindo os do próprio framework CAP e de pacotes reutilizados.  
Use com cuidado.

# Apêndice: Textos de Mensagens Padrão do CAP

Estes são os textos padrão usados pelo runtime do CAP através do bundle `cds.i18n.messages`. Você pode fornecer suas próprias traduções para essas chaves em seus arquivos `\_i18n/messages\_<locale>.properties`.

## Listá de Mensagens de Validação

MULTIPLE\_ERRORS = Multiple errors occurred, see details below.  
ASSERT\_FORMAT = Enter a value matching the pattern {1}.  
ASSERT\_RANGE = Enter a value between {1} and {2}.  
ASSERT\_ENUM = Enter one of the allowed values: {1}.  
ASSERT\_MANDATORY = Provide the missing value.

## Códigos de Status HTTP Traduzíveis

400 = Bad Request	414 = URI Too Long
401 = Unauthorized	415 = Unsupported Media Type
403 = Forbidden	416 = Range Not Satisfiable
404 = Not Found	417 = Expectation Failed
405 = Method Not Allowed	422 = Unprocessable Content
406 = Not Acceptable	424 = Failed Dependency
407 = Proxy Authentication Required	428 = Precondition Required
408 = Request Timeout	429 = Too Many Requests
409 = Conflict	431 = Request Header Fields Too Large
410 = Gone	451 = Unavailable For Legal Reasons
411 = Length Required	500 = Internal Server Error
412 = Precondition Failed	501 = Not Implemented
413 = Payload Too Large	502 = Bad Gateway
	503 = Service Unavailable
	504 = Gateway Timeout