

[<< Back to Dashboard](#)

Internationalization (i18n) & Resource Bundles

Contents

- [Internationalization \(i18n\) & Resource Bundles](#)
 - [Introduction](#)
 - [Credits](#)
 - [Configuration](#)
 - [Coding for i18n](#)
 - [Resource Bundle Usage](#)
 - [String Substitutions](#)
 - [Resource Bundle Tools](#)
 - [How do I change Locales?](#)
 - [Notes](#)
 - [Best Practices](#)
 - [Resources](#)

Covers up to version 3.5.0

Introduction

ColdBox can provide your applications with internationalization and localization capabilities out of the box. We also can provide you with file based language resources in your applications. A project can be internationalized and use resource bundles for displaying per locale strings or just use internationalization without resource bundles. All of these capabilities are provided by the core framework and two awesome plugins: *i18n*, *ResourceBundle*. The ColdBox bundle [download](#) also includes several i18N sample applications that you can learn from.

Credits

These plugins were based on Mr. Paul Hastings work <http://www.sustainablegis.com/blog/cfg11n/>

Configuration

You have a **i18n** configuration structure in your [ConfigurationCFC](#) that you will use to set the default language for each user, where locale information is stored, what resource bundle to load and more.

Key	Type	Required	Default	Description
defaultResourceBundle	relative path	false	---	The base path of where resource bundles will be stored and the initial name of the default resource bundle to load. Ex: <i>includes/i18n/main</i> means that the name of the resource bundles will start with <i>main_language_countrycode</i> .
defaultLocale	string	false	---	The default locale to give users when hitting your application. Please remember that this is using standard java locale syntax. Some examples are <i>es_SV</i> , <i>es_DO</i> . This value gets appended to the default resource bundle setting to define the default language property file to load: <i>includes/main_en_US.properties</i> . So make sure your files are in the following format: {name}_{java standard locale}.properties
localeStorage	string	false	session	This is where the framework will store the client's locale in order to track them and you have three possibilities: <ul style="list-style-type: none"> • session • client • cookie • request
unknownTranslation	numeric	false	---	A setting used by the resource bundle plugin whenever a translation cannot be found. Instead of returning the default bogus <i>_UNKNOWN_</i> , you can choose your own string to return.

```
//i18n & Localization
i18n = {
  defaultResourceBundle = "includes/i18n/main" ,
  defaultLocale = "en_US" ,
  localeStorage = "cookie" ,
  unknownTranslation = "***NOT FOUND**"
};
```

The code above will tell the framework to load the resource bundle **main** for locale **en_US** and save the user locale information in the **cookie** scope. **The only available user scopes are client, session and cookie.** You can also use the config declaration for a i18n project with no resource bundles by using the following declaration:

```
//i18n & Localization
i18n = {
  defaultLocale = "en_es" ,
  localeStorage = "cookie"
};
```

You eliminate the resource bundle element or leave it blank. The framework will not load the bundle.

From this point, the framework loads any default resource bundle into the ColdBox cache and sets the locale you will be using. It is then up to you (DEVELOPER) to code the necessary hooks for internationalization. The last element is the

UnknownTranslation element that is used in the resource bundles plugin. If a resource is asked for that does not exist in the bundle, this element contents will be used or if not declared, the default of **UNKOWN_TRANSLATION** will be used.

Important: All language resource bundles are stored in the configuration structure of your application and are lazy loaded in. So if a language is not used, then it does not get loaded. This is separate to where the user's locale information is stored.

Coding for i18n

Now that you have completed the initialization of the i18n tags in the configuration file, you are ready to code for them. There are several methods that are available to all handlers, plugins, interceptors, layouts and views that deal with i18n via the super type class. You can use these methods directly or go to the i18n plugin directly.

- **getfwLocale()** - Get the current user's locale
- **setfwLocale(locale)** - Set the current user's locale
- **getResource(resource,[default],[locale])** - Get a language resource from the default user's locale or a specific locale.

Your locale is #getFwLocale()#!

```
// localized button
#html.submitButton(value=getResource( 'btn.submit' ))#

// function change a user's locale
function changeLocale(event,rc,prc){
    setFwLocale( rc.locale );
    setNextEvent( 'home' );
}
```

Note: Please note that the i18N plugin gets cached by default on the ColdBox Cache indefinitely. You do not need to do any persistence on this object.

A common practice for localized applications is to store a reference to the *i18n* plugin object in the *prc* scope for easy access throughout your application so you are not constantly calling *getPlugin("i18n")* all over the place.

```
function onRequestStart(event,rc,prc){
    prc.i18n = getPlugin( "i18n" );
}
```

You can then use this utility for i18n specific methods, below is a simple example:

```
<cfoutput >
<strong>Locale: </strong> <br />#prc.i18n.getfwLocale()# <br />
<strong>Language: </strong> <br />#prc.i18n.showLanguage()# <br />
<strong>Country: </strong> <br />#prc.i18n.showCountry()# <br />
<strong>TimeZone: </strong> <br />#prc.i18n.getServerTZ()# <br />
<strong>i18nDateFormat: </strong> <br />#prc.i18n.i18nDateFormat(prc.i18n.toEpoch(now()),1)# <br />
<strong>i18nTimeFormat: </strong> <br />#prc.i18n.i18nTimeFormat(prc.i18n.toEpoch(now()),2)# <br />
<hr>
<strong>I18NUtilVersion: </strong> <br />#prc.i18n.getVersion().I18NUtilVersion# <br>
<strong>I18NUtilDate: </strong> <br />#prc.i18n.dateLocaleFormat(prc.i18n.getVersion().I18NUtilDate)# <br>
<strong>Java version: </strong> <br />#prc.i18n.getVersion().javaVersion# <br>
</cfoutput>
```

Resource Bundle Usage

You have declared the i18n section in your config and now you want to display locale specific text on your layouts, views, event handlers, etc. Well, all you need to do is use the **getResource("Key")** method. This method can be called from anywhere within the framework to retrieve a key from the current set locale's resource bundle. If you have not used resource bundles before or have no clue of what they are, please see the [Resources](#) at the end of this guide. It is a basic concept of getting key values from a structure.

String Substitutions

I recommend that you read the [API](#) for the resource bundle and the i18 plugin as they contain tons of methods that you can use for your internationalization and localization needs. One such method that is very handy is the **formatRBString()** method. This method is used in conjunction with the **getResource()** method. This will allow you to do dynamic text substitutions within resource strings. Wow! That was a mouthful. What does it mean? Well, it means that if you have a resource that looks like this:

```
confirmMessage=Thank you {1} for your recent order. Please call {2} for any suggestions.
```

Then you can actually substitute the **{1}** and **{2}** with dynamic data very easily using this **formatRBString()** method.

- **formatRBString(string rbString, any substituteValues)**

This method takes in a resource string and substitution values as a simple value or an array of values. So to substitute values, I would do something like:

```
<cfset oRB = getPlugin( "resourceBundle" ) >
<cfset orderDetails = ArrayNew(1) >
<cfset orderDetails[1] = "Luis Majano" >
<cfset orderDetails[2] = "1-800-555-5555" >

<cfoutput> #oRB.formatRBString(getResource( 'confirmMessage' ),orderDetails)# </cfoutput>
```

Resource Bundle Tools

- A great utility that I have found is [Attesoro](#), which can help you build your resource bundles. It is also included with ColdBox
- Eclipse Resource Bundle Editor Plugin: <http://sourceforge.net/projects/eclipse-rbe/> is another FANTASTIC editor for eclipse.
- IBM's [Resource Bundle Manager](#) which is awesome too.

Below you can see some screenshots of the eclipse resource bundle editor plugin. I really recommend this tool also

```
<div align="center" >



</div>
```

Note: Make sure your files are all utf-8 encoding. It's also good i18n practice to liberally use `cfprocessingdirective`

Here is a simple example of how to use the `getResource()` method:

```
<p??></p>
<h2>#getResource( "about" )# ColdBox </h2>
<p>#getResource( "aboutcoldbox" )# </p>
</div>
```

It makes a call to the `getResource` method with a key to retrieve. ColdBox then retrieves the appropriate key from the loaded resource bundle. If no key is found it will return an `_UNKNOWNTRANSLATION` value.

How do I change Locales?

Well, it would not make any sense to use i18n for just one locale, would it? That is why you need to be able to change the framework's locale programmatically. You can do this in two ways:

1. Using the i18n plugin and calling the `setfwLocale()` method.
2. Using the i18n plugin instance that you loaded into the application scope (or any scope you loaded it in) and calling the `setfwLocale` method on it.

Below you can see an example taken from the samples gallery:

```
<cffunction name="doChangeLocale" access="public" returntype="void" output="false">
    <cfargument name="event" type="any">
    <!-- Change Locale -->
    <cfset getPlugin( "i18n" ).setfwLocale(event.getValue( "locale" ))>
    <cfset event.setNextEvent( 'main.home' )>
</cffunction>
```

It calls the plugin instance and tells it to load the locale submitted. If an incorrect or unsupported locale (by the JVM) then the framework will throw an invalid locale exception.

Notes

The i18n plugin contains several methods that you can use for i18n purposes, from language codes, timezone support, offsets, etc. You also have the `resourceBundle` plugin that you can use to load on the fly resource bundle files, parse them, etc. Please see the `cfc` documentation for all the possibilities with these plugins.

Best Practices

- eclipse (not CFEclipse) doesn't add a BOM to UTF-8 encoded files.
- Always use `cfprocessingdirective`

```
<cfprocessingdirective pageencoding="utf-8">
```

- ColdBox uses the core java resource bundle flavor so you have to use a proper resource bundle tool to manage these.

Resources

- <http://www.melody-soft.com/html/unifier.html> (vista-compatibleUnifier converts a batch of plain text or HTML files in various characters set encoding to Unicode in UTF-16 or UTF-8 encoding)
- <http://www.i18ngurus.com/>
- <http://www.xencraft.com/resources/webi18ntutorial.pdf>
- http://www.adobe.com/support/coldfusion/internationalization/internationalization_cfm/
- <http://coldfusion.sys-con.com/read/43795.htm>
- <http://sourceforge.net/projects/eclipse-rbe/>
- [Attesoro](#) : A great Resource Bundles utility.
- [rbManager from icu4j](#)
- [RBman](#)