Article
Vitaliy Serdtsev · May 20, 2017 10m read

Localization in Caché DBMS

This is a translation of the following article. Thanks @Evgeny Shvarov for the help in translation.

Let's assume that you wrote a program that shows "Hello World!", for example:

```
  write "Hello, World!"
```

The program works and everyone is happy.

With time, however, your program becomes more complex, gets more features and you eventually need to show the same string in different languages. Moreover you don't know the number and names of these languages.

The spoiler below contains a description of how the task of multi-language localization is solved in Caché.

Brief overview

The Caché DBMS comes with a ready mechanism that facilitates the localization of string resources in console programs, web application interfaces, JavaScript files, error messages and such.

> Note: We took a cursory look at this topic in one of our previous articles.

For example, we have a project with multiple classes, programs, web pages, JS scripts and so forth.

The localization mechanism works in the following way:

1. during **compilation**, all the strings to be localized are extracted an saved to the database in a particular format.
2. in the **compiled** code, these strings are replaced with special placeholders that are filled with corresponding string values (depending on the language selected) at **runtime**.

The localization process is fully transparent for the developer.

The developer does not need to manually populate a string container (database table or a resource file) or write any code to manage this whole infrastructure, including the following: change of language at runtime, data export/import into various formats for translators, etc.

Here's what we end up with:

1. readable and uncluttered source code;
2. automatically populated storage of strings being translated;

Note: When strings are removed from the code, they are not removed from the storage. To clear the storage from such phantoms, it's easier to empty it completely and recompile the entire project.

3. possibility to change the current language "on the go". This applies both to web applications and regular programs;
4. possibility to get a string in a specific language from a particular domain (we'll cover domains a bit later in this article);
5. ready methods for exporting/importing the storage to XML.

All right, let's now have a closer look at how it works and analyze a few examples.

Introduction

Let's create a MAC program with the following content:

```
#Include %occMessages
test() {

  write "$$$DefaultLanguage=",$$$DefaultLanguage,!
  write "$$$SessionLanguage=",$$$SessionLanguage,!

  set msg1=$$$Text("??????, ???!","asd")
  set msg2=$$$Text("@my@??????, ???!","asd")
  write msg1,!,msg2,!

}
```

Result:

```
USER>d ^test
$$$DefaultLanguage=ru
$$$SessionLanguage=ru
??????, ???!
??????, ???!
```

So what do we get as the result?

First of all, we have a new global in the DB

```
^CacheMsg("asd") = "ru"
^CacheMsg("asd","ru",2915927081) = "??????, ???!"
^CacheMsg("asd","ru","my") = "??????, ???!"
```

Then, if you place the cursor over the $$$Text macro, you will see the code it unfolds into.

For example, the intermediate (unfolded) program code (INT-code) will look like this:

```
test() {
  write "$$$DefaultLanguage=",$get(^%SYS("LANGUAGE","CURRENT"),"en"),!
  write "$$$SessionLanguage=",$get(^||%Language,"en"),!
  set msg1=$get(^CacheMsg("asd",$get(^||%Language,"en"),"2915927081"),"??????, ???!")
  set msg2=$get(^CacheMsg("asd",$get(^||%Language,"en"),"my"),"??????, ???!")
  write msg1,!,msg2,!
}
```

As for the example above, please note the following:

1. strings in the program should be originally written in the language that is specified by default in the current locale of the Caché DBMS.

    Note: If you are using string identifiers instead of their hash, this is not that important.

2. for each string, the macros calculates its CRC32, and all data - CRC32 or string identifier, domain, current system language - is saved to the **^CacheMsg** global;
3. instead of the string, the system inserts some code that takes account of the value in the private **^||%Language** global;
4. if the user requests a string in a language for which no translation exists (no data in the storage), the original string will be returned;
5. the domains mechanism allows you to logically split strings being localized - for example, different translations of the same strings and such.

If for some reason you don't like the current algorithm of the **$$$Text** macro - for instance, you want to set the default language in a different way or store data in a different place - you can create your own version of it.

To do this, use the ##Expression and/or ##Function macro.

Let's continue with our example.

Let's add a new language. To do this, export the string storage to a file and give it to translators, then import the translation back in, this time with a different language.

Data can be exported in many ways and in different formats.

We will use the standard methods of the %MessageDictionary class: **Import**(), **ImportDir**(), **Export**(), **ExportDomainList**():

```
  do ##class(%MessageDictionary).Export("messages.xml","ru")
```

We'll get a new file called "*messages_ru.xml* " in our DB folder. Let's rename it to "*messages_en.xml* ", change its language to "**en**" and translate its content.

Once done, import it back to our storage:

```
  do ##class(%MessageDictionary).Import("messages_en.xml")
```

The global will look like this:

```
^CacheMsg("asd") = "ru"
^CacheMsg("asd","en",2915927081) = "Hello, World!"
^CacheMsg("asd","en","my") = "Hello, World!"
^CacheMsg("asd","ru",2915927081) = "??????, ???!"
^CacheMsg("asd","ru","my") = "??????, ???!"
```

We can now change the language "on the go", like this:

```
#Include %occMessages

test()
{

  set $$$SessionLanguageNode="ru"

  set msg1=$$$Text("??????, ???!","asd")
  set msg2=$$$Text("@my@??????, ???!","asd")
  write msg1,!,msg2,!

  set $$$SessionLanguageNode="en"

  set msg1=$$$Text("??????, ???!","asd")
  set msg2=$$$Text("@my@??????, ???!","asd")
  write msg1,!,msg2,!

  set $$$SessionLanguageNode="pt-br"

  set msg1=$$$Text("??????, ???!","asd")
  set msg2=$$$Text("@my@??????, ???!","asd")
  write msg1,!,msg2,!

}
```

Result:

```
USER>d ^test
??????, ???!
??????, ???!
Hello, World!
Hello, World!
??????, ???!
??????, ???!
```

Take a look at the last variant.

Example of localizing a non-web application (a regular class)

Localization of class methods:

```
Include %occErrors

Class demo.test Extends %Persistent
{

Parameter DOMAIN = "asd";

ClassMethod Test()
{
  do ##class(%MessageDictionary).SetSessionLanguage("ru")

  write $$$Text("??????, ???!"),!
  do ##class(%MessageDictionary).SetSessionLanguage("en")

  write $$$Text("??????, ???!"),!

  do ##class(%MessageDictionary).SetSessionLanguage("pt-br")

  write $$$Text("??????, ???!"),!

  #dim ex as %Exception.AbstractException

  try
  {

    $$$ThrowStatus($$$ERR($$$AccessDenied))

  }catch (ex)
  {
    write $system.Status.GetErrorText(ex.AsStatus(),"ru"),!
    write $system.Status.GetErrorText(ex.AsStatus(),"en"),!
    write $system.Status.GetErrorText(ex.AsStatus(),"pt-br"),!
  }
}

}
```

Note: Apparently, you can use the macros described above.

Result:

```
USER>d ##class(demo.test).Test()
??????, ???!
Hello, World!
??????, ???!
?????? #822: ???????? ? ???????
ERROR #822: Access Denied
ERRO #822: Acesso Negado
```

Pay attention to the following nuances:

- exception messages have already been translated into several languages. Since these are system messages, their data is stored in the system **%qCacheMsg** global;
- we defined the name of the domain just once, since the **$$$Text** macro is intended for being used in classes;
- although the **$$$Text** macro was written for web applications, it still works for an offline environment, too.

Example of a web application localization

Let's take a look at th following example:

```
/// Created using the page template: Default
Class demo.test Extends %ZEN.Component.page
{

/// Name of the application that this page belongs to.
Parameter APPLICATION;

/// displayed name for a new application.
Parameter PAGENAME;

/// Domain used for localization.
Parameter DOMAIN = "asd";

/// This  Style contains a definition of the page's CSS style.
XData Style
{
<style type="text/css">
</style>
}

/// This XML block describes the content of this page.
XData Contents [ XMLNamespace = "http://www.intersystems.com/zen" ]
{
<page xmlns="http://www.intersystems.com/zen" title="">
  <checkbox onchange="zenPage.ChangeLanguage();"/>
  <button caption="Client" onclick="zenPage.clientTest(2,3);"/>
  <button caption="Server" onclick="zenAlert(zenPage.ServerTest(1,2));"/>
</page>
}

ClientMethod clientTest(
  a,
  b) [ Language = javascript ]
{
  zenAlert(
          $$$FormatText($$$Text("Result(1)^ %$# @*&' %1=%2"),'"',a+b),'\n',
          zenText('msg3',a+b),'\n',
          $$$Text("Hello from the browser!")
          );
}

ClassMethod ServerTest(
  A,
  B) As %String [ ZenMethod ]
{
  &js<zenAlert(#(..QuoteJS($$$FormatText($$$Text(
"Result(2)^ %$# @*&' ""=%1"),A+B)))#);>
  quit $$$TextJS("Hello from Caché!")
}

Method ChangeLanguage() [ ZenMethod ]
{
  #dim %session as %CSP.Session
  set %session.Language=$select(%session.Language="en":"ru",1:"en")
  &js<zenPage.gotoPage(#(..QuoteJS(..Link($classname()_".cls")))#);>
}

Method %OnGetJSResources(ByRef pResources As %String) As %Status [ Private ]
{
  Set pResources("msg3") = $$$Text("Result(3)^ %$# @*&' ""=%1")
  Quit $$$OK
}

}
```

There are some novelties worth mentioning:

1. there are two options for localizing messages on the client side:
   - using the *$$$Text* method defined in the "*zenutils.js*" file;
   - using a combination of the **zenText()** method on the client side and the server-side method **%OnGetJSResources()**

   More information is available in the official documentation: Localization for Client Side Text
2. Some attributes of ZEN components support localization initially: various headings, tips, etc.

   If you need to create your own object-oriented components - based, for instance, on jQuery or extJS, or built from scratch, - you can use a special data type called **%ZEN.Datatype.caption**: Localization for Zen Components
3. to change a language, you can use the **Language** property of the **%session** and/or **%response** objects: Zen Special Variables

Initially, the session uses the language set in the browser:

Creation of a custom error message dictionary

The tools we reviewed above should suffice to do this.

However, there is an integrated method that helps automate this process a bit.

Let's proceed.

Let's create a "*messagesru.xml* " file containing error messages with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<MsgFile Language="ru">
  <MsgDomain Domain="asd">
    <Message Id="-1" Name="ErrorName1">????????? ? ????? ?????? 1</Message>
    <Message Id="-2" Name="ErrorName2">????????? ? ????? ?????? 2 %1 %2</Message>
  </MsgDomain>
</MsgFile>
```

Let's import it to the DB:

```
do ##class(%MessageDictionary).Import("messages_ru.xml")
```

Two globals were created in the database:

- ^CacheMsg

  ```
  USER>zw ^CacheMsg
  ^CacheMsg("asd","ru",-2)="????????? ? ????? ?????? 2 %1 %2"
  ^CacheMsg("asd","ru",-1)="????????? ? ????? ?????? 1"
  ```

- ^CacheMsgNames

  ```
  USER>zw ^CacheMsgNames
  ^CacheMsgNames("asd",-2)="ErrorName2"
  ^CacheMsgNames("asd",-1)="ErrorName1"
  ```

Generating an Include file called "CustomErrors":

```
USER>Do ##class(%MessageDictionary).GenerateInclude("CustomErrors",,"asd",1)

Generating CustomErrors.INC ...
```

> Note: More details are available in the official documentation for the GenerateInclude()method.

File "*CustomErrors.inc*":

```
#define asdErrorName2 "<asd>-2"
#define asdErrorName1 "<asd>-1"
```

We can now use error codes and/or short error names, for example:

```
Include CustomErrors

Class demo.test [ Abstract ]
{

ClassMethod test(A As %Integer) As %Status
{
  if A=1 Quit $$$ERROR($$$asdErrorName1)
  if A=2 Quit $$$ERROR($$$asdErrorName2,"f","6")
  Quit $$$OK
}

}
```

Results:

```
USER>d $system.OBJ.DisplayError(##class(demo.test).test(1))
```

```
?????? -1: ????????? ? ????? ?????? 1
USER>d $system.OBJ.DisplayError(##class(demo.test).test(2))

?????? -2: ????????? ? ????? ?????? 2 f 6

USER>w $system.Status.GetErrorText(##class(demo.test).test(1),"en")
ERROR -1: Message about some error 1

USER>w $system.Status.GetErrorText(##class(demo.test).test(2),"en")
ERROR -2: Message about some error 2 f 6
```

Note: Messages for the English language were created in an identical manner.

#Compiler #XML #ObjectScript #Localization #Caché

Source URL:https://community.intersystems.com/post/localization-cach%C3%A9-dbms