
# String Localization and Message Dictionaries

Version 2018.1
2024-11-07

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

*String Localization and Message Dictionaries*
PDF generated on 2024-11-07
InterSystems Caché® Version 2018.1
Copyright © 2024 InterSystems Corporation
All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble®, InterSystems HealthShare®, InterSystems IRIS®, and TrakCare are registered trademarks of InterSystems Corporation. HealthShare® CMS Solution Pack™, HealthShare® Health Connect Cloud™, InterSystems® Data Fabric Studio™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, and InterSystems TotalView™ For Asset Management are trademarks of InterSystems Corporation. TrakCare is a registered trademark in Australia and the European Union.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

**InterSystems Worldwide Response Center (WRC)**
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

# Table of Contents

# String Localization and Message Dictionaries

This article explains:

- How Caché supports string localization
- The structure of a message dictionary
- The structure of an XML message file
- How to import, export, and otherwise manage a message dictionary

## 1 Overview

When you *localize* the text for an application, you create an inventory of text strings in one language, then establish a convention for substituting translated versions of these messages in another language when the application locale is different.

Caché supports the following process for localizing strings:

1. Developers include localizable strings within their code (within a REST service, a CSP or Zen application, or a DeepSee model).

   The mechanisms for this vary, but the most common mechanism is the `$$$Text` macro. In the place of a hardcoded literal string, the developer includes an instance of the `$$$Text` macro, providing values for the macro arguments as follows:

   - The default string
   - The domain to which this string belongs (localization is easier to manage when the strings are grouped into domains)
   - The language code of the default string

   For example, instead of this:

   ```
   write "Hello world"
   ```

   The developer includes this:

   ```
   write $$$TEXT("Hello world","sampledomain","en-us")
   ```

2. When the code is compiled, the compiler generates entries in the message dictionary for each unique instance of the `$$$Text` macro.

   The message dictionary is a global and so can be easily viewed (for example) in the Management Portal. Caché provides class methods to help with common tasks.

3. When development is complete, release engineers export the message dictionary for that domain or for all domains.

   The result is one or more XML message files that contain the text strings in the original language.

4. Release engineers send these files to translators, requesting translated versions.

It may be helpful to also send the CacheMessages.dtd file, which describes the schema required by the message files. The file format, however, is quite simple and the DTD may not be necessary.

5. Translators work with the XML message file using any XML authoring tool they prefer. Essentially they translate text from the original language to a new language, without changing the surrounding XML.
6. Translators return a new XML message file that:
   - Conforms to CacheMessages.dtd
   - Identifies a new RFC1766 value for the *language* attribute of the `<MsgFile>` element.
   - Contains translated text in the identified language.
7. Release engineers import the translated XML message files into the same namespace from which the original was exported.

   Translated and original texts coexist in the message dictionary.
8. At runtime, the application chooses which text to display based on the browser default language.

For details on steps 1 and 2, see the following chapters:

- "Localizing a REST Service" in the chapter "Creating REST Services" in *Creating REST Services in Caché*
- "Localizing Text in a CSP Application" in *Using Caché Server Pages (CSP)*
- "Zen Localization" in *Developing Zen Applications*
- "Performing Localization" in the *DeepSee Implementation Guide*.

The remainder of this article discusses managing the message dictionary.

# 2 Message Dictionaries

A message dictionary is a global that contains text strings organized by domain name, language name, and message ID:

- The *text* of each message is a string of up to 32K characters. The string may be longer if the database has long strings enabled, but the default maximum is 32K. A message may consist solely of text, or it may also contain one or more parameters specified by *%1*, *%2*, etc. Caché allows you to replace these parameters with text (such as a file name within an error message) when the application page needs to display the message.
- A *domain* name is any arbitrary string. It identifies a group of related text items, such as all messages for a specific application or page. If you assign a domain to a set of messages, you can later ask Caché to perform a particular operation on all messages with the same domain.

  A domain name is case-sensitive and may contain upper- and lowercase characters. If a domain name begins with %, Caché considers all of the messages in that domain to be system messages that are visible in all namespaces. Otherwise, when you create a message it is visible only in the namespace in which it is defined.
- A *language* name is an all-lowercase language tag that conforms to RFC1766. It consists of one or more parts: a primary language tag (such as `en` or `ja`) optionally followed by a hyphen (-) and a secondary language tag (`en-gb` or `ja-jp`).
- A *message ID* is any arbitrary string; it uniquely identifies a message. The message ID only needs to be unique within a domain. You may assign a message ID or allow Caché to assign one, depending on the conventions you use to create the message. A message ID is case-sensitive and may contain upper- and lowercase characters.

## 2.1 Message Dictionary Storage

Each user-defined namespace stores its message dictionary in a subscripted global called ^*CacheMsg*. The order of subscripts in ^*CacheMsg* is domain, language, and message ID.

To view ^*CacheMsg* for a namespace:

1. Start the Management Portal.
2. Switch to the namespace of interest.
3. Click **System Explorer** > **Globals**.
4. In the **CacheMsg** row, click **View**.

The following excerpt is taken from the Caché SAMPLES namespace. It shows the entries for two languages (English and Spanish) in a localization domain called `sample`. Where a line is too long to fit in this display, this example truncates the line at right:

```
^CacheMsg("sample","en","LangComment") = "Demo of displaying a page in the local
^CacheMsg("sample","en","LangDesc") = "This sample demonstrates how to display a
^CacheMsg("sample","en","LangDisplay") = "Display Resource"
^CacheMsg("sample","en","LangEnglish") = "English"
^CacheMsg("sample","en","LangSet") = "Set Language"
^CacheMsg("sample","en","LangText1") = $c(13,10)_"This page has been translated
^CacheMsg("sample","en","LangText2") = $c(13,10)_"To display this page in a lang
^CacheMsg("sample","en","LangText3") = $c(13,10)_"To display the XML resource fi
^CacheMsg("sample","en","LangText4") = $c(13,10)_"The source code of this page s
^CacheMsg("sample","en","LangTitle") = "Language Localization Example"
^CacheMsg("sample","en","Language") = "English"
^CacheMsg("sample","en","menu") = "Samples Menu"
^CacheMsg("sample","en","source") = "Source"
^CacheMsg("sample","es","LangComment") = "Demo de presentación de una p
^CacheMsg("sample","es","LangDesc") = "Este ejemplo muestra como presentar una p
^CacheMsg("sample","es","LangDisplay") = "Mostrar Recurso"
^CacheMsg("sample","es","LangEnglish") = "Spanish"
^CacheMsg("sample","es","LangSet") = "Establecer Lenguaje"
^CacheMsg("sample","es","LangText1") = $c(13,10)_"Esta página ha sido tra
^CacheMsg("sample","es","LangText2") = $c(13,10)_"Para mostrar esta págin
^CacheMsg("sample","es","LangText3") = $c(13,10)_"Para mostrar el fichero fuente
^CacheMsg("sample","es","LangText4") = $c(13,10)_"El código fuente de esta
^CacheMsg("sample","es","LangTitle") = "Ejemplo de Localización de Lenguaj
^CacheMsg("sample","es","Language") = "Español"
^CacheMsg("sample","es","menu") = "Menú Samples"
^CacheMsg("sample","es","source") = "Fuente"
```

# 3 XML Message Files

An *XML message file* is an export of message dictionary. This is also the required format for any messages that you wish to import into Caché. (For export and import instructions, see the section "Managing a Message Dictionary.")

An XML message file must conform to the following DTD:

*InstallDir*`\dev\csp\rules\CacheMessages.dtd`

Where *InstallDir* is the Caché installation directory.

Whenever possible, XML message files should use UTF-8 encoding. However, in certain cases a developer or translator might use local platform encodings, such as `shift-jis`, for ease of editing the XML message file. Whatever encoding is used for the XML file, it must be supported by the Caché locale for the application, and it must be able to express the messages for the language.

An XML message file may contain messages for one language and multiple domains.

## 3.1 <MsgFile> Element

The <MsgFile> element is the top-level container for the XML message file, and there is only one <MsgFile> element per file.

The <MsgFile> element has one required attribute, *Language*. The value of the <MsgFile> *Language* attribute is an all-lowercase RFC1766 code that identifies the language for the file. It consists of one or more parts: a primary language tag (such as en or ja) optionally followed by a hyphen (-) and a secondary language tag (en-gb or ja-jp).

In the following example, this language is "en" (English).

#### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<MsgFile Language="en">
    <MsgDomain Domain="sample">
        <Message Id="source">Source</Message>
        <Message Id="menu">Samples Menu</Message>
    </MsgDomain>
</MsgFile>
```

<MsgFile> must contain at least one <MsgDomain> element. It may contain more than one <MsgDomain>.

## 3.2 <MsgDomain> Element

The <MsgDomain> element has one required attribute, *Domain*. The value of the <MsgDomain> *Domain* attribute is one of the domain names that you are using to organize the messages in your application.

Any <MsgDomain> element may contain zero or more <Message> elements.

## 3.3 <Message> Element

The <Message> element has one required attribute, *Id*. The value of the <Message> *Id* attribute is one of the message ID strings that you are using to organize the messages in your application.

Any <Message> element may contain a text string. The string may consist of any of the following items, separately or in combination:

- Simple text, as permitted by the file format
- Substitution arguments %1, %2, %3, or %4
- HTML formatting
- A string expression in ObjectScript format

The following example uses %1, %2, the HTML tag for bold formatting, and the ObjectScript string convention that two successive double quote characters indicate a single double quote:

#### XML

```
<Message>
  The session $Username="&lt;b&gt;%1&lt;/b&gt;" $Roles="&lt;b>%2&lt;/b&gt;"
</Message>
```

# 4 Managing a Message Dictionary

This section summarizes the %Library.MessageDictionary methods that are most commonly used when working with a message dictionary. You can use these methods to:

- Import messages from an XML message file
- Export messages to an XML message file
- Delete messages from a message dictionary
- List messages in a message dictionary

## 4.1 Importing an XML Message File

To import an XML message file, open the Terminal and do the following:

1. Change to the namespace where you are developing the application:

```
ZN "myNamespace"
```

2. Run the import command. By default, each language is in a separate XML message file, with the locale name at the end of the filename. Thus:
   - You can import only those messages in a particular language:

```
SET file="C:\myLocation\Messages_ja-jp.xml"
DO ##class(%Library.MessageDictionary).Import(file)
```

   - Or, import several languages for the same application:

```
SET myFiles="C:\myLocation"
DO ##class(%Library.MessageDictionary).ImportDir(myFiles,"d")
```

3. Examine the *^CacheMsg* global in the same namespace to see the result.

The following topics summarize both import methods.

### 4.1.1 Importing a Specific XML Message File

The %Library.MessageDictionary class method **Import()** has the following signature:

```
classmethod Import(filepath As %String, flag As %String = "") returns %Status
```

| Argument | Description |
|---|---|
| *filepath* | Import the XML message file specified by *filepath*. Make sure that only XML message files are in the directory as other XML files generate errors. |
| *flag* | (Optional) If provided, the d flag (display) indicates that the Terminal console will display confirmation messages as files are imported. Otherwise, there is no confirmation. |

### 4.1.2 Importing All the XML Message Files in a Directory

The %Library.MessageDictionary class method **ImportDir()** has the following signature:

```
classmethod ImportDir(directory As %String, flag As %String = "") returns %Status
```

| Argument | Description |
|---|---|
| *directory* | Import all of the XML message files in the specified directory. |
| *flag* | (Optional) If provided, the d flag (display) indicates that the Terminal console will display confirmation messages as files are imported. Otherwise, there is no confirmation. |

## 4.2 Exporting an XML Message File

To export portions of a message dictionary to an XML message file, do the following within the Terminal:

1. Change to the namespace where you are developing the application:

```
ZN "myNamespace"
```

2. Identify the output file and its location:

```
SET file="C:\myLocation\Messages.xml"
```

3. Run the export command:
   - It may be practical to export only those messages in a particular domain:

```
DO ##class(%Library.MessageDictionary).ExportDomainList(file,"myDomain")
```

   - Or, to export all the messages in the namespace:

```
DO ##class(%Library.MessageDictionary).Export(file)
```

The following topics summarize both export methods.

### 4.2.1 Exporting Specific Domains in One Language

The %Library.MessageDictionary class method **ExportDomainList()** has the following signature:

```
classmethod ExportDomainList(file As %String, domainList As %String, language As %String) returns
%Status
```

| Argument | Description |
|---|---|
| *file* | (Required) A template for the output filename in this format:<br>*filepath*.*ext*<br>Caché names the output file by appending the *language* value to the *filepath* with an extension of *ext*. |
| *domainList* | (Optional) A comma-separated list of domains to be exported. |
| *language* | (Optional) Only the specified *language* is exported. The value must be an all-lowercase RFC1766 code. If not provided, the value defaults to the system default language, a value that is stored in the special variable `$$$DefaultLanguage`. |

### 4.2.2 Exporting All Domains in Specific Languages

The %Library.MessageDictionary class method **Export()** has the following signature:

```
classmethod Export(file As %String, languages As %String = "", flag As %String = "") returns %Status
```

| Argument | Description |
|---|---|
| *file* | (Required) A template for the output filename in this format:<br>*filepath*.*ext*<br>The name of the output file is *filepathlanguage-code*.*ext*<br>For example, if *file* is `c:/temp/mylang_.txt` and *languages* includes the language code `ja-jp`, then one of the output files is named `c:/temp/mylang_ja-jp.txt` |
| *languages* | (Optional) A comma-separated list of language codes. Each value in the list must be an all-lowercase RFC1766 code. If *languages* is not specified, or is empty, all languages in the database are exported. Each language is exported to a separate file using the conventions described for the *file* argument. |
| *flag* | (Optional) If provided, the `s` flag (system) indicates that system message dictionaries are to be exported in addition to application message dictionaries. Otherwise, only application message dictionaries are exported. |

## 4.3 Deleting Messages

To delete messages use the following command:

```
Set status = ##class(%MessageDictionary).Delete(languages,flag)
```

*languages* is an optional comma-separated list of languages. If *languages* is not specified, all languages are deleted. The default value is to delete application messages only. The `s` flag (system) is an optional flag indicating whether to also delete system messages. The message names associated with include files are always deleted, but not the include files. The `d` flag (display) is also supported.

## 4.4 Listing Messages

To get the list of all languages that have messages loaded for a specified domain, use the **GetLanguages()** method:

```
Set list = ##class(%MessageDictionary).GetLanguages(domain,flag)
```

**GetLanguages()** returns a %ListofDateTypes format list of language codes in the standard RFC1766 format and all in lowercase. If *domain* is specified, then only languages that exist for the specified domain are included in the list. Otherwise, all languages are included in the list. The `s` flag (system) is an optional flag indicating whether languages supported by system or application messages are to be returned. The default value is to return the languages for application messages. The `d` flag (display) is also supported.

# 5 See Also

- "Localizing Text in a CSP Application" in *Using Caché Server Pages (CSP)*
- "Zen Localization" in *Developing Zen Applications*
- "Performing Localization" in the *DeepSee Implementation Guide*