String Localization and Message Dictionaries

Version 2023.1
2024-07-11

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

String Localization and Message Dictionaries
InterSystems IRIS Data Platform Version 2023.1 2024-07-11

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

1

Introduction to String Localization

This page provides an overview of string localization in InterSystems products — a key part of the localization support.

1.1 Localization Basics

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. Accordingly, InterSystems products support the following process for localizing strings:

1. Developers include localizable strings within their code (within a web application or a Business Intelligence 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:

   ObjectScript

   ```
   write "Hello world"
   ```

   The developer includes this:

   ObjectScript

   ```
   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. There are class methods to manage the data.

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 file that contain the text strings in the original language.

4. Release engineers send these files to translators, requesting translated versions.
5. Translators work with the files 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 file that has the same structure and that:
    - 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.

1.2 Localization Macros

This section gives details on the macros used for localization. These macros are contained in %occMessages.inc, which is included in %occInclude.inc):

- `$$$Text` returns a string
- `$$$TextJS` returns a string that is escaped for use in JavaScript
- `$$$TextHTML` returns a string that is escaped for use in HTML

Each of these macros takes three arguments: the default string, the domain to which this string belongs, and the language code of the default string. When code is compiled, the compiler generates entries in the message dictionary for each unique set of values of the arguments.

1.3 Macro Details

The `$$$Text`, `$$$TextJS`, and `$$$TextHTML` macros take the following arguments in order:

text

A non-empty literal string. The format used for *text* may be:

"*actualText*"

Or:

"*@textId@actualText*"

Where *textId* is a message ID and *actualText* is the text of the message.

The string *actualText* 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

- An ObjectScript string expression

If provided, the *textId* is used as the message ID. If `@`*textId*`@` is not specified, the system generates a new *textId* by calculating the 32–bit CRC (Cyclic Redundancy Check) of this text. If the *textId* is specified and a message already exists with this ID, the existing message is checked to see if it has the same text as *actualText*. If not, an error is reported.

domain

(Optional) String specifying the domain for the new message. If not specified, *domain* defaults to the value of the *DOMAIN* class parameter at compile time and *%response.Domain* at runtime.

language

(Optional) RFC1766 code specifying the language. InterSystems IRIS converts this string to all-lowercase. If not specified, *language* defaults as follows:

- At compile time: `$$$DefaultLanguage`.
- At runtime (within a CSP-based web application), the default is *%response.Language*, or if no value is defined for *%response.Language* then `$$$DefaultLanguage`.
- At runtime (in other contexts) : `$$$DefaultLanguage`.

1.4 Message Arguments and $$$FormatText Macros

If the message text contains arguments (*%1*, *%2*, *%3*, *%4*) you must specify the corresponding substitution text before displaying the text. Because `$$$Text` returns a string, you can use any string operation native to your coding language. For example, in JavaScript:

```
var prompt = '#($$$TextHTML("Remove user %1 from this Role?"))#';
prompt = prompt.replace(/%1/g,member.userName);
```

InterSystems provides additional macros that enable you to substitute text for message arguments:

- `$$$FormatText`
- `$$$FormatTextJS` (applies JavaScript escaping to the $$$FormatText result)
- `$$$FormatTextHTML` (applies HTML escaping to the $$$FormatText result)

These macros are in %occMessages.inc, which is included in %occInclude.inc. These macros take the following arguments in order:

text

The message text, a string. Use the value returned by $$$Text.

arg1, *arg2*, and so on

Substitution text for the message arguments.

1.5 See Also

- Introduction to CSP-Based Web Applications
- System Macros
- Message Dictionaries
- XML Message Files
- Managing a Message Dictionary

2

Message Dictionaries

String localization uses message dictionaries.

2.1 Organization of a Message Dictionary

The term *message dictionary* refers to a global (or part of a global) that stores localizable messages. Each namespace has a message dictionary. The global 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. You can 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 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 `%`, InterSystems 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 the compiler 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.

2.2 Message Dictionary Storage

The *message dictionary* global is called *^IRIS.Msg*, and its subscripts are (in order) domain, language, and message ID. The value of each node is the text associated with that domain, language, and message ID.

To view *^IRIS.Msg* for a namespace:

1. Start the Management Portal.
2. Switch to the namespace of interest.

3. Click **System Explorer** > **Globals**.
4. In the **IRIS.Msg** row, click **View**.

For example:

```
^IRIS.Msg("mydomain")="en"
^IRIS.Msg("mydomain","en",338288369)="sample string"
^IRIS.Msg("mydomain","en",3486914925)="another sample string"
```

2.3 See Also

- Introduction to String Localization
- XML Message Files
- Managing a Message Dictionary

3

XML Message Files

An *XML message file* is an export of a message dictionary, which is used in string localization. This is also the required format for any messages that you wish to import. (For export and import instructions, see Managing a Message Dictionary.)

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 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>
```

3.4 See Also

- Introduction to String Localization
- Message Dictionaries
- Managing a Message Dictionary

4

Managing a Message Dictionary

This topic summarizes the %Library.MessageDictionary methods that are most commonly used when working with a message dictionary, used in string localization.

4.1 Importing Messages

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

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

   ObjectScript

   ```
   set $namespace = "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:

     ObjectScript

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

   - Or, import several languages for the same application:

     ObjectScript

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

3. Examine the ^*IRIS.Msg* 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
```

Where:

- *filepath* specifies the full pathname of the message file.
- *flag* is an optional flag. 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
```

Where

- *filepath* specifies a directory file. Make sure that only XML message files are in the directory as other XML files generate errors.
- *flag* is an optional flag. 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 Messages

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:

   ObjectScript

   ```
   set $namespace = "myNamespace"
   ```

2. Identify the output file and its location:

   ObjectScript

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

3. Run the export command:

   - It may be practical to export only those messages in a particular domain:

     ObjectScript

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

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

     ObjectScript

     ```
     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
```

Where:

- *file* is a template for the output filename in this format:

  filepath`.`*ext*

  The actual output file name appends the *language* value to the *filepath* with an extension of *ext*.
- *domainList* is an optional comma-separated list of domains to be exported.
- *language* is an all-lowercase RFC1766 code. If not provided, the value defaults to the system default language.

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
```

Where:

- *file* is a template for the output filename in this format:

  filepath`.`*ext*

  The name of the output file is *filepathlanguage-code*`.`*ext*

  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* is an optional 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* is an optional flag. 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:

ObjectScript

```
 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:

ObjectScript

```
 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.

4.5 See Also

- Introduction to String Localization
- Message Dictionaries
- XML Message Files