NAME

msgcat − Tcl message catalog

SYNOPSIS

package require Tcl 8.5

package require msgcat 1.4.2

::msgcat::mc *src-string* ?*arg arg ...*?

::msgcat::mcmax ?*src-string src-string ...*?

::msgcat::mclocale ?*newLocale*?

::msgcat::mcpreferences

::msgcat::mcload *dirname*

::msgcat::mcset *locale src-string* ?*translate-string*?

::msgcat::mcmset *locale src-trans-list*

::msgcat::mcunknown *locale src-string*

DESCRIPTION

The **msgcat** package provides a set of functions that can be used to manage multi-lingual user interfaces. Text strings are defined in a “message catalog” which is independent from the application, and which can be edited or localized without modifying the application source code. New languages or locales are provided by adding a new file to the message catalog.

Use of the message catalog is optional by any application or package, but is encouraged if the application or package wishes to be enabled for multi-lingual applications.

COMMANDS

::msgcat::mc *src-string* ?*arg arg ...*?

Returns a translation of *src-string* according to the user’s current locale. If additional arguments past *src-string* are given, the **format** command is used to substitute the additional arguments in the translation of *src-string*.

::msgcat::mc will search the messages defined in the current namespace for a translation of *src-string*; if none is found, it will search in the parent of the current namespace, and so on until it reaches the global namespace. If no translation string exists, **::msgcat::mcunknown** is called and the string returned from **::msgcat::mcunknown** is returned.

::msgcat::mc is the main function used to localize an application. Instead of using an English string directly, an application can pass the English string through **::msgcat::mc** and use the result. If an application is written for a single language in this fashion, then it is easy to add support for additional languages later simply by defining new message catalog entries.

::msgcat::mcmax ?*src-string src-string ...*?

Given several source strings, **::msgcat::mcmax** returns the length of the longest translated string. This is useful when designing localized GUIs, which may require that all buttons, for example, be a fixed width (which will be the width of the widest button).

::msgcat::mclocale ?*newLocale*?

This function sets the locale to *newLocale*. If *newLocale* is omitted, the current locale is returned, otherwise the current locale is set to *newLocale*. msgcat stores and compares the locale in a case-insensitive manner, and returns locales in lowercase. The initial locale is determined by the locale specified in the user’s environment. See **LOCALE SPECIFICATION** below for a description of the locale string format.

::msgcat::mcpreferences
Returns an ordered list of the locales preferred by the user, based on the user's language specification. The list is ordered from most specific to least preference. The list is derived from the current locale set in msgcat by **::msgcat::mclocale**, and cannot be set independently. For example, if the current locale is en_US_funky, then **::msgcat::mcpreferences** returns **{en_US_funky en_US en {}}**.

::msgcat::mcload *dirname*
Searches the specified directory for files that match the language specifications returned by **::msgcat::mcpreferences** (note that these are all lowercase), extended by the file extension ".msg". Each matching file is read in order, assuming a UTF-8 encoding. The file contents are then evaluated as a Tcl script. This means that Unicode characters may be present in the message file either directly in their UTF-8 encoded form, or by use of the backslash-u quoting recognized by Tcl evaluation. The number of message files which matched the specification and were loaded is returned.

::msgcat::mcset *locale src-string* ?*translate-string*?
Sets the translation for *src-string* to *translate-string* in the specified *locale* and the current namespace. If *translate-string* is not specified, *src-string* is used for both. The function returns *translate-string*.

::msgcat::mcmset *locale src-trans-list*
Sets the translation for multiple source strings in *src-trans-list* in the specified *locale* and the current namespace. *src-trans-list* must have an even number of elements and is in the form {*src-string translate-string* ?*src-string translate-string ...*?} **::msgcat::mcmset** can be significantly faster than multiple invocations of **::msgcat::mcset**. The function returns the number of translations set.

::msgcat::mcunknown *locale src-string*
This routine is called by **::msgcat::mc** in the case when a translation for *src-string* is not defined in the current locale. The default action is to return *src-string*. This procedure can be redefined by the application, for example to log error messages for each unknown string. The **::msgcat::mcunknown** procedure is invoked at the same stack context as the call to **::msgcat::mc**. The return value of **::msgcat::mcunknown** is used as the return value for the call to **::msgcat::mc**.

LOCALE SPECIFICATION

The locale is specified to **msgcat** by a locale string passed to **::msgcat::mclocale**. The locale string consists of a language code, an optional country code, and an optional system-specific code, each separated by "_". The country and language codes are specified in standards ISO-639 and ISO-3166. For example, the locale "en" specifies English and "en_US" specifies U.S. English.

When the msgcat package is first loaded, the locale is initialized according to the user's environment. The variables **env(LC_ALL)**, **env(LC_MESSAGES)**, and **env(LANG)** are examined in order. The first of them to have a non-empty value is used to determine the initial locale. The value is parsed according to the XPG4 pattern

```
language[_country][.codeset][@modifier]
```

to extract its parts. The initial locale is then set by calling **::msgcat::mclocale** with the argument

```
language[_country][_modifier]
```

On Windows, if none of those environment variables is set, msgcat will attempt to extract locale information from the registry. If all these attempts to discover an initial locale from the user's environment fail, msgcat defaults to an initial locale of "C".

When a locale is specified by the user, a "best match" search is performed during string translation. For example, if a user specifies en_GB_Funky, the locales "en_GB_Funky", "en_GB", "en" and (the empty string) are searched in order until a matching translation string is found. If no translation string is available, then **::msgcat::mcunknown** is called.

NAMESPACES AND MESSAGE CATALOGS

Strings stored in the message catalog are stored relative to the namespace from which they were added. This allows multiple packages to use the same strings without fear of collisions with other packages. It also allows the source string to be shorter and less prone to typographical error.

For example, executing the code

```
::msgcat::mcset en hello "hello from ::"
namespace eval foo {
   ::msgcat::mcset en hello "hello from ::foo"
}
puts [::msgcat::mc hello]
namespace eval foo {puts [::msgcat::mc hello]}
```

will print

```
hello from ::
hello from ::foo
```

When searching for a translation of a message, the message catalog will search first the current namespace, then the parent of the current namespace, and so on until the global namespace is reached. This allows child namespaces to “inherit” messages from their parent namespace.

For example, executing (in the “en” locale) the code

```
::msgcat::mcset en m1 ":: message1"
::msgcat::mcset en m2 ":: message2"
::msgcat::mcset en m3 ":: message3"
namespace eval ::foo {
   ::msgcat::mcset en m2 "::foo message2"
   ::msgcat::mcset en m3 "::foo message3"
}
namespace eval ::foo::bar {
   ::msgcat::mcset en m3 "::foo::bar message3"
}
namespace import ::msgcat::mc
puts "[mc m1]; [mc m2]; [mc m3]"
namespace eval ::foo {puts "[mc m1]; [mc m2]; [mc m3]"}
namespace eval ::foo::bar {puts "[mc m1]; [mc m2]; [mc m3]"}
```

will print

```
:: message1; :: message2; :: message3
:: message1; ::foo message2; ::foo message3
:: message1; ::foo message2; ::foo::bar message3
```

LOCATION AND FORMAT OF MESSAGE FILES

Message files can be located in any directory, subject to the following conditions:

[1] All message files for a package are in the same directory.

[2] The message file name is a msgcat locale specifier (all lowercase) followed by “.msg”. For example:
es.msg — spanish
en_gb.msg — United Kingdom English

Exception: The message file for the root locale is called “**ROOT.msg**”. This exception is made so as not to cause peculiar behavior, such as marking the message file as “hidden” on Unix file systems.

[3] The file contains a series of calls to **mcset** and **mcmset**, setting the necessary translation strings for the language, likely enclosed in a **namespace eval** so that all source strings are tied to the namespace of the package. For example, a short **es.msg** might contain:

```
namespace eval ::mypackage {
   ::msgcat::mcset es "Free Beer!" "Cerveza Gracias!"
}
```

RECOMMENDED MESSAGE SETUP FOR PACKAGES

If a package is installed into a subdirectory of the **tcl_pkgPath** and loaded via **package require**, the following procedure is recommended.

[1] During package installation, create a subdirectory **msgs** under your package directory.

[2] Copy your *.msg files into that directory.

[3]

Add the following command to your package initialization script:

```
# load language files, stored in msgs subdirectory
::msgcat::mcload [file join [file dirname [info script]] msgs]
```

POSITIONAL CODES FOR FORMAT AND SCAN COMMANDS

It is possible that a message string used as an argument to **format** might have positionally dependent parameters that might need to be repositioned. For example, it might be syntactically desirable to rearrange the sentence structure while translating.

```
format "We produced %d units in location %s" $num $city
format "In location %s we produced %d units" $city $num
```

This can be handled by using the positional parameters:

```
format "We produced %1\$d units in location %2\$s" $num $city
format "In location %2\$s we produced %1\$d units" $num $city
```

Similarly, positional parameters can be used with **scan** to extract values from internationalized strings.

CREDITS

The message catalog code was developed by Mark Harrison.

SEE ALSO

format(3tcl), scan(3tcl), namespace(3tcl), package(3tcl)

KEYWORDS

internationalization, i18n, localization, l10n, message, text, translation