Internationalization with gettext

Supporting multiple natural languages in a user interface

Bastiaan Veelo, PhD.  DConf 2023
Previously…
Previously...

Coming from Pascal: home grown system
  • Translate the source code
Previously...

Coming from Pascal: home grown system
• Translate the source code

Disadvantages:
• Badly scalable. Need to compile for every language.
• Not user friendly. Users only order one or several specific languages, need to change executables to change language.
• Hard to maintain — translation rot / no fallback / no “needs work” markers
• No run-time variations (plural variants)
• No equivalent for “format”:
  • Translators work with fragments, lack context
  • Translators cannot swap arguments
Previously...

Coming from Pascal: home grown system
  • Translate the source code

Disadvantages:
  • Badly scalable. Need to compile for every language.
  • Not user friendly. Users only order one or several specific languages, need to change executables to change language.
  • Hard to maintain — translation rot / no fallback / no “needs work” markers
  • No run-time variations (plural variants)
  • No equivalent for “format”:
    • Translators work with fragments, lack context
    • Translators cannot swap arguments

Transcompilation = opportunity to switch. What is best?
// Vista and later enabled application, this application will not work on OS versions prior to Vista

#include <windows.h>
#include <wchar.h>
#include <strsafe.h>
#include "resource.h"

#define SUFFICIENTLY_LARGE_STRING_BUFFER (MAX_PATH*2)
#define USER_CONFIGURATION_STRING_BUFFER (((LOCALE_NAME_MAX_LENGTH+1)*5)+1)
#define SUFFICIENTLY_LARGE_ERROR_BUFFER (1024*2)

BOOL GetMyUserDefinedLanguages(WCHAR * langStr, DWORD langStrSize);
BOOL ConvertMyLangStrToMultiLangStr(WCHAR * langStr, WCHAR * langMultiStr, DWORD langMultiStrSize);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    UNREFERENCED_PARAMETER(hInstance);
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    UNREFERENCED_PARAMETER(nCmdShow);

    // The following code presents a hypothetical, yet common use pattern of MUI technology
    WCHAR displayBuffer[SUFFICIENTLY_LARGE_ERROR_BUFFER];

    // 1. Application starts by applying any user defined language preferences
    //  (language setting is potentially optional for an application that wishes to strictly use OS system language)
    // 1a. Application looks in pre-defined location for user preferences (registry, file, web, etc.)
    WCHAR userLanguagesString[USER_CONFIGURATION_STRING_BUFFER*2];
    if (!GetMyUserDefinedLanguages(userLanguagesString,USER_CONFIGURATION_STRING_BUFFER*2))
    {
    //...
// Vista and later enabled application, this application will not work on OS versions prior to Vista

#include <windows.h>
#include <wchar.h>
#include <strsafe.h>
#include "resource.h"

#define SUFFICIENTLY_LARGE_STRING_BUFFER (MAX_PATH*2)
#define USER_CONFIGURATION_STRING_BUFFER ((((LOCALE_NAME_MAX_LENGTH+1)*5)+1)
#define SUFFICIENTLY_LARGE_ERROR_BUFFER (1024*2)

BOOL GetMyUserDefinedLanguages(WCHAR * langStr, DWORD langStrSize);
BOOL ConvertMyLangStrToMultiLangStr(WCHAR * langStr, WCHAR * langMultiStr, DWORD langMultiStrSize);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    UNREFERENCED_PARAMETER(hInstance);
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    UNREFERENCED_PARAMETER(nCmdShow);

    // The following code presents a hypothetical, yet common use pattern of MUI technology
    WCHAR displayBuffer[SUFFICIENTLY_LARGE_ERROR_BUFFER];

    // 1. Application starts by applying any user defined language preferences
    // (language setting is potentially optional for an application that wishes to strictly use OS system
    // 1a. Application looks in pre-defined location for user preferences (registry, file, web, etc.)
    WCHAR userLanguagesString[USER_CONFIGURATION_STRING_BUFFER*2];
    if(!GetMyUserDefinedLanguages(userLanguagesString,USER_CONFIGURATION_STRING_BUFFER*2)) {
        swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,"L"FAILURE: Unable to find the user defined language configuration, last error = %d.",GetLastError());
    }
include <windows.h>
#include <wchar.h>
#include <strsafe.h>
#include "resource.h"

#define SUFFICIENTLY_LARGE_STRING_BUFFER (MAX_PATH*2)
#define USER_CONFIGURATION_STRING_BUFFER (((LOCALE_NAME_MAX_LENGTH+1)*5)+1)
#define SUFFICIENTLY_LARGE_ERROR_BUFFER (1024*2)

BOOL GetMyUserDefinedLanguages(WCHAR * langStr, DWORD langStrSize);
BOOL ConvertMyLangStrToMultiLangStr(WCHAR * langStr, WCHAR * langMultiStr, DWORD langMultiStrSize);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    UNREFERENCED_PARAMETER(hInstance);
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
    UNREFERENCED_PARAMETER(nCmdShow);

    // The following code presents a hypothetical, yet common use pattern of MUI technology
    WCHAR displayBuffer[SUFFICIENTLY_LARGE_ERROR_BUFFER];

    // 1. Application starts by applying any user defined language preferences
    // (language setting is potentially optional for an application that wishes to strictly use OS system
    // 1a. Application looks in pre-defined location for user preferences (registry, file, web, etc.)
    WCHAR userLanguagesString[USER_CONFIGURATION_STRING_BUFFER*2];
    if (!GetMyUserDefinedLanguages(userLanguagesString,USER_CONFIGURATION_STRING_BUFFER*2))
    {
        swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to find the user defined language configuration, last error = %d.

    // 1a. Application looks in pre-defined location for user preferences (registry, file, web, etc.)
WCHAR * langMultiStrPtr = langMultiStr;
WCHAR * last = langStr + (langStr[0] == 0xFEFF ? 1 : 0);
WCHAR * context = last;
WCHAR * next = wcstok_s(last, L";, :", &context);
while (next && rtnVal)
{
    // make sure you validate the user input
    if (SUCCEEDED(StringCchLengthW(last, LOCALE_NAME_MAX_LENGTH, &strLen)) && IsValidLocaleName(next))
    {
        langMultiStrPtr[0] = L'\0';
        rtnVal &= SUCCEEDED(StringCchCatW(langMultiStrPtr, (langMultiStrSize - (langMultiStrPtr - langMultiStr)), next));
        langMultiStrPtr += strLen + 1;
    }
    next = wcstok_s(NULL, L";, :", &context);
    if (next)
        last = next;
}
if (rtnVal && (langMultiStrSize - (langMultiStrPtr - langMultiStr))) // make sure there is a double null term for the multi-string
{
    langMultiStrPtr[0] = L'\0';
}
else // fail and guard anyone whom might use the multi-string
{
    langMultiStr[0] = L'\0';
    langMultiStr[1] = L'\0';
}
return rtnVal;
WCHAR * langMultiStrPtr = langMultiStr;
WCHAR * last = langStr + (langStr[0] == 0xFEFF ? 1 : 0);
WCHAR * context = last;
WCHAR * next = wcstok_s(last, L"; :", &context);
while (next && rtnVal)
{
    // make sure you validate the user input
    if (SUCCEEDED(StringCchLengthW(last, LOCALE_NAME_MAX_LENGTH, &strLen)) && IsValidLocaleName(next))
    {
        langMultiStrPtr[0] = L'\0';
        rtnVal &= SUCCEEDED(StringCchCatW(langMultiStrPtr, (langMultiStrSize - (langMultiStrPtr - langMultiStr)), next));
        langMultiStrPtr += strLen + 1;
    }
    next = wcstok_s(NULL, L"; :", &context);
    if (next)
    {
        last = next;
    }
}
if (rtnVal && (langMultiStrSize - (langMultiStrPtr - langMultiStr))) // make sure there is a double null term for the multi-string
{
    langMultiStrPtr[0] = L'\0';
}
else // fail and guard anyone whom might use the multi-string
{
    langMultiStr[0] = L'\0';
    langMultiStr[1] = L'\0';
}
return rtnVal;
// LoadLibraryEx is the preferred alternative for resource modules as used below because it provides increased security and performance over that of LoadLibrary

HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH, NULL, LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);

if (!resContainer)
{
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to load the resource container module, last error = %d." , GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    return 1; // exit
}

// 3. Application parses the resource container to find the appropriate item

WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];

if (LoadStringW(resContainer, HELLO_MUI_STR_0, szHello, SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
{
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to load the resource string, last error = %d." , GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}

// 4. Application presents the discovered resource to the user via UI

swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"%s MUI" , szHello);
MessageBoxW(NULL, displayBuffer, L"HelloMUI", MB_OK | MB_ICONINFORMATION);

// 5. Application cleans up memory associated with the resource container after this item is no longer needed.

if (!FreeLibrary(resContainer))
{
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to unload the resource container, last error = %d." , GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    return 1; // exit
}

return 0;
LoadLibraryEx is the preferred alternative for resource modules as used below because it provides increased security and performance over that of LoadLibrary.

```c
HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH, NULL, LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);
```

If `resContainer` is not loaded correctly:

```c
swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to load the resource container module, last error = %d.", GetLastError());
MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
return 1; // exit
```

3. Application parses the resource container to find the appropriate item.

```c
WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if (LoadStringW(resContainer, HELLO_MUI_STR_0, szHello, SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
```

If the resource string cannot be loaded:

```c
swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to load the resource string, last error = %d.", GetLastError());
MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
FreeLibrary(resContainer);
return 1; // exit
```

4. Application presents the discovered resource to the user via UI.

```c
swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"%s MUI", szHello);
MessageBoxW(NULL, displayBuffer, L"HelloMUI", MB_OK | MB_ICONINFORMATION);
```

5. Application cleans up memory associated with the resource container after this item is no longer needed.

```c
if (!FreeLibrary(resContainer))
```

If the resource container cannot be unloaded correctly:

```c
swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to unload the resource container, last error = %d.", GetLastError());
MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
return 1; // exit
```
// LoadLibraryEx is the preferred alternative for resource modules as used below because it
// provides increased security and performance over that of LoadLibrary

HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH,NULL,LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);
if(!resContainer)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource container module, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}

// 3. Application parses the resource container to find the appropriate item
WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if(LoadStringW(resContainer,HELLO_MUI_STR_0,szHello,SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource string, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}

// 4. Application presents the discovered resource to the user via UI
swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"%s MUI",szHello);
MessageBoxW(NULL,displayBuffer,L"HelloMUI",MB_OK | MB_ICONINFORMATION);

// 5. Application cleans up memory associated with the resource container after this item is no longer needed.
if(!FreeLibrary(resContainer))
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to unload the resource container, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}
LoadLibraryEx is the preferred alternative for resource modules as used below because it provides increased security and performance over that of LoadLibrary.

```c
HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH, NULL, LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);
```

If `!resContainer`:

```c
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to load the resource container module, last error = %d.", GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    return 1; // exit
}
```

3. Application parses the resource container to find the appropriate item.

```c
WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if(LoadStringW(resContainer, HELLO_MUI_STR_0, szHello, SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
{
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to load the resource string, last error = %d.", GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}
```

4. Application presents the discovered resource to the user via UI.

```c
swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"%s MUI", szHello);
MessageBoxW(NULL, displayBuffer, L"HelloMUI", MB_OK | MB_ICONINFORMATION);
```

5. Application cleans up memory associated with the resource container after this item is no longer needed.

```c
if(!FreeLibrary(resContainer))
{
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to unload the resource container, last error = %d.", GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    return 1; // exit
}

return 0;
```
// LoadLibraryEx is the preferred alternative for resource modules as used below because it
// provides increased security and performance over that of LoadLibrary
HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH,NULL,LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);
if(!resContainer)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource container module, last error = %d." ,GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}

// 3. Application parses the resource container to find the appropriate item
WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if(LoadStringW(resContainer,HELLO_MUI_STR_0,szHello,SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource string, last error = %d." ,GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}

// 4. Application presents the discovered resource to the user via UI
swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"%s MUI",szHello);
MessageBoxW(NULL,displayBuffer,L"HelloMUI",MB_OK | MB_ICONINFORMATION);

// 5. Application cleans up memory associated with the resource container after this item is no longer needed.
if(!FreeLibrary(resContainer))
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to unload the resource container module, last error = %d." ,GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}
// LoadLibraryEx is the preferred alternative for resource modules as used below because it
// provides increased security and performance over that of LoadLibrary

HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH,NULL,LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);
if(!resContainer)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource container module, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}

// 3. Application parses the resource container to find the appropriate item

WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if(LoadStringW(resContainer,HELLO_MUI_STR_0,szHello,SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource string, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}

// 4. Application presents the discovered resource to the user via UI

swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"%s MUI",szHello);
MessageBoxW(NULL,displayBuffer,L"HelloMUI",MB_OK | MB_ICONINFORMATION);

// 5. Application cleans up memory associated with the resource container after this item is no longer needed.
if(!FreeLibrary(resContainer))
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to unload the resource container, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}

return 0;
LoadLibraryEx is the preferred alternative for resource modules as used below because it provides increased security and performance over that of LoadLibrary.

```c
HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH, NULL, LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);

if (!resContainer)
{
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to load the resource container module, last error = %d.", GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    return 1; // exit
}
```

3. Application parses the resource container to find the appropriate item.

```c
WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if (LoadStringW(resContainer, HELLO_MUI_STR_0, szHello, SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
{
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to load the resource string, last error = %d.", GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}
```

4. Application presents the discovered resource to the user via UI.

```c
swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"%s MUI", szHello);
MessageBoxW(NULL, displayBuffer, L"HelloMUI", MB_OK | MB_ICONINFORMATION);
```

5. Application cleans up memory associated with the resource container after this item is no longer needed.

```c
if (!FreeLibrary(resContainer))
{
    swprintf_s(displayBuffer, SUFFICIENTLY_LARGE_ERROR_BUFFER, L"FAILURE: Unable to unload the resource container, last error = %d.", GetLastError());
    MessageBoxW(NULL, displayBuffer, L"HelloMUI ERROR!", MB_OK | MB_ICONERROR);
    return 1; // exit
}
```
// provides increased security and performance over that of LoadLibrary
HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH,NULL,LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);
if(!resContainer)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource container module, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}

// 3. Application parses the resource container to find the appropriate item
WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if(LoadStringW(resContainer,HELLO_MUI_STR_0,szHello,SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource string, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}

// 4. Application presents the discovered resource to the user via UI
swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"%s MUI",szHello);
MessageBoxW(NULL,displayBuffer,L"HelloMUI",MB_OK | MB_ICONINFORMATION);

// 5. Application cleans up memory associated with the resource container after this item is no longer needed.
if(!FreeLibrary(resContainer))
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to unload the resource container, last error = %d.",GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}
LoadLibraryEx is the preferred alternative for resource modules as used below because it provides increased security and performance over that of LoadLibrary:

```c
HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH,NULL,LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);
if(!resContainer) 
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource container module, last error = %d." ,GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}
```

3. Application parses the resource container to find the appropriate item:

```c
WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if(LoadStringW(resContainer,HELLO_MUI_STR_0,szHello,SUFFICIENTLY_LARGE_STRING_BUFFER) == 0) 
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource string, last error = %d." ,GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}
```

4. Application presents the discovered resource to the user via UI:

```c
swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"%s MUI",szHello);
MessageBoxW(NULL,displayBuffer,L"HelloMUI",MB_OK | MB_ICONINFORMATION);
```

5. Application cleans up memory associated with the resource container after this item is no longer needed:

```c
if(!FreeLibrary(resContainer)) 
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to unload the resource container, last error = %d." ,GetLastError());
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}
```
// LoadLibraryEx is the preferred alternative for resource modules as used below because it
// provides increased security and performance over that of LoadLibrary
HMODULE resContainer = LoadLibraryExW(HELLO_MODULE_CONTRIVED_FILE_PATH,NULL,LOAD_LIBRARY_AS_IMAGE_RESOURCE | LOAD_LIBRARY_AS_DATAFILE);
if(!resContainer)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource container module, last error = %d.
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}

// 3. Application parses the resource container to find the appropriate item
WCHAR szHello[SUFFICIENTLY_LARGE_STRING_BUFFER];
if(LoadStringW(resContainer,HELLO_MUI_STR_0,szHello,SUFFICIENTLY_LARGE_STRING_BUFFER) == 0)
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to load the resource string, last error = %d.
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    FreeLibrary(resContainer);
    return 1; // exit
}

// 4. Application presents the discovered resource to the user via UI
swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"%s MUI",szHello);
MessageBoxW(NULL,displayBuffer,L"HelloMUI",MB_OK | MB_ICONINFORMATION);

// 5. Application cleans up memory associated with the resource container after this item is no longer needed.
if(!FreeLibrary(resContainer))
{
    swprintf_s(displayBuffer,SUFFICIENTLY_LARGE_ERROR_BUFFER,L"FAILURE: Unable to unload the resource container module, last error = %d.
    MessageBoxW(NULL,displayBuffer,L"HelloMUI ERROR!",MB_OK | MB_ICONERROR);
    return 1; // exit
}
import java.util.*;

public class I18NSample {

    static public void main(String[] args) {

        String language;
        String country;

        if (args.length != 2) {
            language = new String("en");
            country = new String("US");
        } else {
            language = new String(args[0]);
            country = new String(args[1]);
        }

        Locale currentLocale;
        ResourceBundle messages;

        currentLocale = new Locale(language, country);

        messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
        System.out.println(messages.getString("greetings"));
        System.out.println(messages.getString("inquiry"));
        System.out.println(messages.getString("farewell"));
    }
}
import java.util.*;

public class I18NSample {
    static public void main(String[] args) {
        String language;
        String country;

        if (args.length != 2) {
            language = new String("en");
            country = new String("US");
        } else {
            language = new String(args[0]);
            country = new String(args[1]);
        }

        Locale currentLocale;
        ResourceBundle messages;

        currentLocale = new Locale(language, country);

        messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
        System.out.println(messages.getString("greetings"));
        System.out.println(messages.getString("inquiry"));
        System.out.println(messages.getString("farewell"));
    }
}
LoginWidget::LoginWidget()
{
    QLabel *label = new QLabel(tr("Password:"));
    ...
}
LoginWidget::LoginWidget()
{
    QLabel *label = new QLabel(tr("Password:"));
    ... 
}

void FileCopier::showProgress(int done, int total, const QString &currentFile)
{
    label.setText(tr("%1 of %2 files copied.\nCopying: %3").arg(done).arg(total).arg(currentFile));
}
LoginWidget::LoginWidget()
{
    QLabel *label = new QLabel(tr("Password:"));
    ...
}

void FileCopier::showProgress(int done, int total, const QString &currentFile)
{
    label.setText(tr("%1 of %2 files copied.\nCopying: %3").arg(done).arg(total).arg(currentFile));
}

MyWindow::MyWindow()
{
    QLabel *senderLabel = new QLabel(tr("Name:"));
    QLabel *recipientLabel = new QLabel(tr("Name:" , "recipient"));
    ...
}
LoginWidget::LoginWidget()
{
    QLabel *label = new QLabel(tr("Password:"));
    ...  
}

void FileCopier::showProgress(int done, int total, const QString &currentFile)
{
    label.setText(tr("%1 of %2 files copied.\nCopying: %3").arg(done).arg(total).arg(currentFile));
}

MyWindow::MyWindow()
{
    QLabel *senderLabel = new QLabel(tr("Name:"));
    QLabel *recipientLabel = new QLabel(tr("Name:"),"recipient");
    ...  
}

showMessage(tr("%n message(s) saved", "", messages.count()));
LoginWidget::LoginWidget()
{
    QLabel *label = new QLabel(tr("Password:"));
    ...
}

void FileCopier::showProgress(int done, int total, const QString &currentFile)
{
    label.setText(tr("%1 of %2 files copied.\nCopying: %3").arg(done).arg(total).arg(currentFile));
}

MyWindow::MyWindow()
{
    QLabel *senderLabel = new QLabel(tr("Name:"));
    QLabel *recipientLabel = new QLabel(tr("Name:"), "recipient");
    ...
}

showMessage(tr("%n message(s) saved", ",", messages.count()));

//: This name refers to a host name.
hostNameLabel->setText(tr("Name:"));
In computing, gettext is an internationalization and localization (i18n and i10n) system commonly used for writing multilingual programs on Unix-like computer operating systems. One of the main benefits of gettext is that it separates programming from translating.[3] The most commonly used implementation of gettext is GNU gettext,[4] released by the GNU Project in 1995. The runtime library is libintl. gettext provides an option to use different strings for any number of plural forms of nouns, but this feature has no support for grammatical gender. The main filename extensions used by this system are .POT (Portable Object Template), .PO (Portable Object) and .MO (Machine Object).[5]

**History**  [ edit ]

Initially, POSIX provided no means of localizing messages. Two proposals were raised in the late 1980s, the 1988 Uniform gettext and the 1989 X/Open catgets (XPG-3 § 5). Sun Microsystems implemented the first gettext in 1993. The Unix and POSIX developers never really agreed on what kind of interface to use (the other option is the X/Open catgets), so many C libraries, including glibc, implemented both.[6] As of August 2019, whether gettext should be part of POSIX was still a point of debate in the Austin Group, despite the fact that its old foe has already fallen out of use. Concerns cited included its dependence on the system-set locale (a global variable subject to multithreading problems) and its support for newer C-language extensions involving wide strings.[7]
Gettext is an internationalization and localization (i18n and l10n) system commonly used for writing multilingual programs on Unix-like computer operating systems. One of the main benefits of gettext is that it separates programming from translating. The most commonly used implementation of gettext is GNU gettext, released by the GNU Project in 1995. The runtime library is libintl. gettext provides an option to use different strings for any number of plural forms of nouns, but this feature has no support for grammatical gender. The main filename extensions used by this system are .PO (Portable Object Template), .PO (Portable Object) and .MO (Machine Object).

History

Initially, POSIX provided no means of localizing messages. Two proposals were raised in the late 1980s, the 1988 Uniforum gettext and the 1989 X/Open catgets (XPG-3 § 5). Sun Microsystems implemented the first gettext in 1993. The Unix and POSIX developers never really agreed on what kind of interface to use (the other option is the X/Open catgets), so many C libraries, including glibc, implemented both. As of August 2019, whether gettext should be part of POSIX was still a point of debate in the Austin Group, despite the fact that its old foe has already fallen out of use. Concerns cited included its dependence on the system-set locale (a global variable subject to multithreading problems) and its support for newer C-language extensions involving wide strings.

<table>
<thead>
<tr>
<th>Original author(s)</th>
<th>Sun Microsystems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer(s)</td>
<td>various</td>
</tr>
<tr>
<td>Initial release</td>
<td>1990; 33 years ago[1]</td>
</tr>
<tr>
<td>Stable release</td>
<td>0.22[2] / 17 June 2023; 2 months ago</td>
</tr>
<tr>
<td>Repository</td>
<td>various based on OpenSolaris and GNU gettext</td>
</tr>
<tr>
<td>Operating system</td>
<td>Cross-platform</td>
</tr>
<tr>
<td>Type</td>
<td>Internationalization and localization</td>
</tr>
<tr>
<td>License</td>
<td>Various free software licenses</td>
</tr>
<tr>
<td>Website</td>
<td><a href="http://www.gnu.org/software/gettext/">www.gnu.org/software/gettext/</a></td>
</tr>
</tbody>
</table>
Initial release 1990; 33 years ago[1]

In computing, gettext is an internationalization and localization (i18n and l10n) system commonly used for writing multilingual programs on Unix-like computer operating systems. One of the main benefits of gettext is that it separates programming from translating.[3] The most commonly used implementation of gettext is GNU gettext,[4] released by the GNU Project in 1995. The runtime library is libintl. gettext provides an option to use different strings for any number of plural forms of nouns, but this feature has no support for grammatical gender. The main filename extensions used by this system are .POT (Portable Object Template), .PO (Portable Object) and .MO (Machine Object).[5]

History[edit]

Initially, POSIX provided no means of localizing messages. Two proposals were raised in the late 1980s: the 1988 X/Open catalog gettext and the 1989 X/Open catalog getcets (XPG4). Neither were implemented. The Unix and POSIX standards and their ISO counterparts did not address what kind of interface to use (the other option was to rely on C libraries, including glibc), implemented both.[6] As of August 1999, whether gettext should be part of POSIX was still a point of debate in the Austin Group, despite the fact that its old foe has already fallen out of use. Concerns included its dependence on the system-set locale (a global variable subject to multithreading problems) and its support for newer C-language extensions involving wide strings.[7]
In computing, gettext is an internationalization and localization (i18n and l10n) system commonly used for writing multilingual programs on Unix-like computer operating systems. One of the main benefits of gettext is that it separates programming from translating.\[^3\] The most commonly used implementation of gettext is GNU gettext,\[^4\] released by the GNU Project in 1995. The runtime library is libintl. gettext provides an option to use different strings for any number of plural forms of nouns, but this feature has no support for grammatical gender. The main filename extensions used by this system are .POT (Portable Object Template), .PO (Portable Object) and .MO (Machine Object).\[^5\]

### History

Initially, POSIX provided no means of localizing messages. Two proposals were raised in the late 1980s: the 1989 X/Open catalog recommended the forum gettext and the 1989 X/Open catalog implemented the XPG4. POSIX implemented the forum gettext and the 1989 X/Open catalog option, but both POSIX and XPG4 also offered various implementations of the C library.

As of August 2019, whether gettext should be part of POSIX was still a point of debate in the Austin Group, despite the fact that its old foe has already fallen out of use. Concerns cited included its dependence on the system-set locale (a global variable subject to multithreading problems) and its support for newer C-language extensions involving wide strings.\[^7\]
GNU gettext, released by the GNU Project in 1995.

History

Initially, POSIX provided no means of localizing messages. Two proposals were raised in the late 1980s: the 1988 implementation of gettext and the 1989 X/Open catalog (XPG4) proposal. The latter implemented the former and was based on what kind of internationalization support was already available in the various C libraries of both. As of August 2019, whether gettext should be part of POSIX was still a point of debate in the Austin Group, despite the fact that its old foe has already fallen out of use. Concerns cited included its dependence on the system-set locale (a global variable subject to multithreading problems) and its support for newer C-language extensions involving wide strings.
In computing, gettext is an internationalization and localization (I18N and L10N) system commonly used for writing multilingual programs on Unix-like computer operating systems. One of the main benefits of gettext is that it separates programming from translating. The most commonly used implementation of gettext is GNU gettext released by the GNU Project in 1995. The runtime library GNU gettext released by the GNU Project in 1995.

History
Initially, POSIX provided no means of localizing messages. Two proposals were raised in the late 1980s, the standard forum gettext and the 1989 X/Open catalog. XPG3 POSIX (XPG3) implemented the first option, but only for C libraries. However, the second option, gettext, has already fallen out of use. As of August 2019, whether gettext should be part of POSIX was still a point of debate in the Austin Group, despite the fact that its old foe has already fallen out of use. Concerns cited included its dependence on the system-set locale (a global variable subject to multithreading problems) and its support for newer C-language extensions involving wide strings.
Initial release 1990; 33 years ago[1]

In computing, gettext is an internationalization and localization (l10n and i18n) system commonly used for writing multilingual programs on Unix-like computer operating systems. One of the main benefits of gettext is that it separates programming from translating. The most commonly used implementation of gettext is GNU gettext[4] released by the GNU Project in 1995. The runtime library


History [edit]

Initially, POSIX provided no means of localizing messages. Two proposals were raised in the late 1980s: the POSIX platform gettext and the 1988 X/Open catalog of C-language options. ([XPG3]) The X/Open catalog of C-language options, which differed on what kinds of locale information was available. As of August 2019, whether gettext should be part of POSIX was still a point of debate in the Austin Group, despite the fact that its old foe has already fallen out of use. Concerns cited included its dependence on the system-set locale (a global variable subject to multithreading problems) and its support for newer C-language extensions involving wide strings.[7]
Utilities

Runtime

GNU Gettext
Utilities
- Extraction

Runtime

GNU Gettext

Third Party
- Parters

Tools
Utilities
- Extraction
- Initialization

Runtime

GNU Gettext

Third Party

Parters

Tools
Utilities
- Extraction
- Initialization
- Maintenance
- Deployment

Runtime
- Lookup
- Selection

GNU Gettext

Third Party

Partners

Tools
Utilities
- Extraction
- Initialization
- Maintenance
- Deployment

Runtime
- Lookup
- Selection

GNU Gettext

Third Party
- Syntax highlighting

Tools

Partners
Utilities
- Extraction
- Initialization
- Maintenance
- Deployment

Runtime
- Lookup
- Selection

GNU Gettext

Third Party
- Syntax highlighting
- Dedicated editors
- Translation APIs

Partners
Utilities
• **Extraction**
• Initialization
• Maintenance
• Deployment

Runtime
• **Lookup**
• **Selection**

GNU Gettext

Tools
• Syntax highlighting
• Dedicated editors
• Translation APIs

Third Party
• Translation services

Partners
#include <libintl.h>

puts(gettext("This is a translatable string.\n"));
printf(gettext("String \"%s\" has %d characters.\n"), s, strlen(s));
#include <libintl.h>

puts(gettext("This is a translatable string.\n"));
printf(gettext("String ""%s"" has %d characters.\n"),
             s, strlen(s));
// TRANSLATORS: Comment regarding following string.
printf(pgettext("Message recipient", "Name: %s"),
              recipient_name);
#include <libintl.h>

puts(gettext("This is a translatable string.\n"));
printf(gettext("String \"%s\" has %d characters.\n"),
    s, strlen(s));
// TRANSLATORS: Comment regarding following string.
printf(pgettext("Message recipient", "Name: %s"),
    recipient_name);
printf(ngettext("One file copied", "%d files copied", n), n);

https://www.gnu.org/software/gettext/manual/
#include <libintl.h>

puts(gettext("This is a translatable string.\n"));
printf(gettext("String "%s" has %d characters.\n"), s, strlen(s));

// TRANSLATORS: Comment regarding following string.
printf(pgettext("Message recipient", "Name: %s"), recipient_name);
printf(ngettext("One file copied", "%d files copied", n), n);

#define gettext_noop(String) String

static const char *messages[] = {
    gettext_noop("some very meaningful message"),
    gettext_noop("and another one")
};

const char *str;
str = index > 1 ?
    gettext("a default message") :
    gettext(messages[index]);
**gettext 1.0.5**

Internationalization compatible with the GNU gettext utilities.

To use this package, run the following command in your project's root directory:

```
dub add gettext
```

**Manual usage**

Put the following dependency into your project's dependences section:

```
dub.json
"gettext": "~>1.0.5"
```

```
dub.sdl
dependency "gettext" version="~>1.0.5"
```

This package provides sub packages which can be used individually:

- `gettext:merge` - Merge existing translations with a new template.
- `gettext:po2mo` - Batch execution of gettext msgfmt.
- `gettext:todo` - Find unmarked string literals.

Registered by Bastiaan Veelo

1.0.5 released a month ago

- veelo/gettext

BSL-1.0

Copyright © 2022, SARC B.V.

Authors:

- Bastiaan Veelo

Sub packages:

- `gettext:merge`
- `gettext:po2mo`
- `gettext:todo`

Dependencies:

- `mojo`

Versions:

- **1.0.5** 2023–Jul–14
- **1.0.4** 2023–Aug–26
- **1.0.3** 2022–Aug–17
- **1.0.2** 2022–Aug–01
- **1.0.1** 2022–Jul–15

Show all 7 versions

Download Stats:

- 0 downloads today
- 2 downloads this week
- 2 downloads this month
- 62 downloads total

The GNU gettext utilities provide a well established solution for the internationalization of software. It allows users to switch between natural languages without switching executable. Many commercial translation
import gettext;

writeln(tr!"This is a translatable string.");
import gettext: _ = tr;

writeln(_!'"This is a translatable string."');
https://code.dlang.org/packages/gettext
writeln(tr!"\tIndented by escape code.");
writeln(tr!"\tIndented by escape code.");
writeln(tr!`Popular "wysiwyg" string.`);
writeln(tr!"\tIndented by escape code.");
writeln(tr!`Popular "wysiwyg" string.`);
writeln(tr!r"Same thing, less popular.");
writeln(tr!"\tIndented by escape code.");
writeln(tr!`Popular "wysiwyg" string.`);
writeln(tr!r"Same thing, less popular.");
writeln(tr!q"<A delimited string>" /* etc.*/)};
writeln(tr!"\tIndented by escape code.");
writeln(tr!`Popular "wysiwyg" string.`);
writeln(tr!r"Same thing, less popular.");
writeln(tr!q"<A delimited string>"); // etc.
writeln(tr!q"EOS
This
is a multi-line
heredoc string
EOS");
writeln(tr!'\tIndented by escape code.');
writeln(tr!'Popular "wysiwyg" string.');
writeln(tr!'r"Same thing, less popular."');
writeln(tr!'q"<A delimited string>"'); // etc.
writeln(tr!'q"EOS
This
is a multi-line
heredoc string
EOS"');
writeln(tr!'q{void foo();}}');
https://code.dlang.org/packages/gettext
// Before:
writefln("%d green bottle(s) hanging on the wall", n);
// Before:
writefln("%d green bottle(s) hanging on the wall", n);
writefln(n == 1 ?
    "One green bottle hanging on the wall" :
    "%d green bottles hanging on the wall", n);
// Before:
writefln("%d green bottle(s) hanging on the wall", n);
writefln(n == 1 ?
    "One green bottle hanging on the wall" :
    "%d green bottles hanging on the wall", n);

// After:
writeln(tr!("one green bottle hanging on the wall",
    "%d green bottles hanging on the wall") (n));
https://code.dlang.org/packages/gettext
// Bad:
writeln(tr!"Welcome " ~ player ~ tr!", you may make a wish);
// Bad:
writeln(tr!"Welcome " ~ player ~ tr!", you may make a wish");

// Good:
writefln(tr!("Welcome %s, you may make a wish"), player);
// Bad:
writeln(tr!"Welcome " ~ player ~ tr!", you may make a wish");

// Good:
writefln(tr!("Welcome %s, you may make a wish"), player);
writefln(tr!("Welcome %s, you may make a wish",
    "Welcome %s, you may make %d wishes")(n), player);
https://code.dlang.org/packages/gettext
foreach (i, where; "hand", "bush")
writefln(i == 0 ? "%d bird in the %s" :
"%d birds in the %s", i + 1, where);
```d
foreach (i, where; ["hand", "bush"])
    writeln(i == 0 ? "%d bird in the %s":
               "%d birds in the %s", i + 1, where);

foreach (i, where; [tr!"hand", tr!"bush"])
    writeln(tr!("One bird in the %1$s",
                "%2$d birds in the %1$s") (i + 1),
             where);
```

https://code.dlang.org/packages/gettext
Format Strings

The functions contained in this package use format strings. A format string describes the layout of another string for reading or writing purposes. A format string is composed of normal text interspersed with format specifiers. A format specifier starts with a percentage sign '%', optionally followed by one or more parameters and ends with a format indicator. A format indicator may be a simple format character or a compound indicator.

Format strings are composed according to the following grammar:

```
FormatString:
  FormatStringItem FormatString
FormatStringItem:
  Character
  FormatSpecifier
FormatSpecifier:
  '%' Parameters FormatIndicator

FormatIndicator:
  FormatCharacter
  CompoundIndicator
FormatCharacter:
  see remark below
CompoundIndicator:
  (' FormatString %')
  (' FormatString %'| Delimiter %')
Delimiter
  empty
  Character Delimiter

Parameters:
  see remark below
```
tr!("Walter Bright", Comment("Proper name. Phonetically: 'wɔltər braɪt'"));
Review the draft.

Context ("nautical")
Comment ("Nautical term! "Draft" = how deep the bottom of the ship is below the water level.")
tr!("Walter Bright", Comment("Proper name. Phonetically: 'wɔltər braɪt'"));

tr!("Review the draft.", Context("document"));
tr!("Review the draft.", Context("nautical"),
    Comment(`Nautical term! "Draft" = how deep the bottom` ~ `of the ship is below the water level.`));

tr!("One license.", "%d licenses.", Context("software"),
    Comment("Notice to translator."))(n);
tr!("One license.", "%d licenses.", Context("driver's"))(n);
static const magic = tr!"Compile time translation?!";
static const magic = tr!"Compile time translation?!";

enum {
    monday = tr!"Monday",
    tuesday = tr!"Tuesday",
    wednesday = tr!"Wednesday",
    thursday = tr!"Thursday",
    friday = tr!"Friday",
    saturday = tr!"Saturday",
    sunday = tr!"Sunday",
}
static const magic = tr!"Compile time translation?!";

type enum { 
  monday = tr!"Monday",
  tuesday = tr!"Tuesday",
  wednesday = tr!"Wednesday",
  thursday = tr!"Thursday",
  friday = tr!"Friday",
  saturday = tr!"Saturday",
  sunday = tr!"Sunday",
}

struct S
{
  auto day = monday;
  auto city = tr!"Gothenburg";
}
#include <libintl.h>

#define gettext noop(String) String

static const char *messages[] = {
    gettext_noop("some very meaningful message"),
    gettext_noop("and another one")
};

const char *str = index > 1 ?
    gettext("a default message") :
    gettext(messages[index]);
#include <libintl.h>
#definetext_noop(String) String

static const char *messages[] = {
    gettext_noop("some very meaningful message"),
    gettext_noop("and another one")
};
const char *str = index > 1 ?
    gettext("a default message") :
    gettext(messages[index]);
mixin(`writeln(tr!"This is mixed in code."));`);
import std.stdio;

void main()
{
    writeln("Edit source/app.d to start your project.");
}

{
    "authors": [
        "Bastiaan Veelo"
    ],
    "copyright": "Copyright © 2023, Bastiaan Veelo",
    "description": "A minimal D application.",
    "license": "proprietary",
    "name": "helloworld"
}
import std.stdio;

void main()
{
    writeln("Edit source/app.d to start your project.");
}

},
    "copyright": "Copyright © 2023, Bastiaan Veelo",
    "description": "A minimal D application.",
    "license": "proprietary",
    "name": "helloworld"}
import std.stdio, gettext;

void main()
{
    writeln(tr!"Edit source/app.d to start your project.");
}

{"authors": ["Bastiaan Veelo"],
  "copyright": "Copyright © 2023, Bastiaan Veelo",
  "description": "A minimal D application.",
  "license": "proprietary",
  "name": "helloworld",
  "dependencies": {
      "gettext": "~>1"
  }"}
import std.stdio, gettext;

void main()
{
    writeln(tr!"Edit source/app.d to start your project.");
}

"license": "proprietary",
"name": "helloworld",
"dependencies": {
    "gettext": "~>1"
}
Downloading

The latest release is 0.22, which can be downloaded from https://ftp.gnu.org/pub/gnu/gettext/gettext-0.22.tar.gz. For other ways to obtain gettext, please read How to get GNU Software.

The latest development sources can be obtained from the savannah project, using Git.

Michele Locati kindly provides precompiled binaries for Windows on his site.

Maintainer

ggettext is currently being maintained by Bruno Haible and Daiki Ueno. Please use the mailing lists for contact.
```plaintext
import std.stdio, gettext;

void main()
{
    mixin(gettext.main);
    writeln(tr ! "Edit source/app.d to start your project.");
}

"dependencies": {
    "gettext": "~>1"
},
"targetType": "executable",
"configurations": [
    {
        "name": "default"
    },
    {
        "name": "i18n",
        "preGenerateCommands": [
            "dub run --config=xgettext",
            "dub run gettext:merge -- --popath=po --backup=none",
            "dub run gettext:po2mo -- --popath=po --mopath=mo"
        ],
        "copyFiles": [
            "mo"
        ]
    },
    {
        "name": "xgettext",
        "targetPath": ".\xgettext",
        "versions": [ "xgettext" ],
        "subConfigurations": {
            "gettext": "xgettext"
        }
    }
]}"
```d
import std.stdio, gettext;

do mixin(gettext.main);

do writeln(!"tr!" "Edit source/app.d to start your project.");
```

```
"dependencies": {
  "gettext": "~>1",
},
"targetType": "executable",
"configurations": [
  {
    "name": "default",
  },
  {
    "name": "i18n",
    "preGenerateCommands": [
      "dub run --config=xgettext",
      "dub run gettext:merge -- --popath=po --backup=none",
      "dub run gettext:po2mo -- --popath=po --mopath=mo",
    ],
    "copyFiles": [
      "mo",
    ],
  },
  {
    "name": "xgettext",
    "targetPath": ".xgettext",
    "versions": [
      "xgettext",
    ],
    "subConfigurations": {
      "gettext": "xgettext",
    },
  }
],

```

```
bastiaan$ dub build --config=i18n

Pre-gen Running commands for helloworld
Starting Performing "debug" build using /Library/D/dmd/bin/dmd for x86_64.
Up-to-date mofile 0.2.1: target for configuration [library] is up to date.
Up-to-date gettext 1.0.7: target for configuration [xgettext] is up to date.
Up-to-date helloworld ~master: target for configuration [xgettext] is up to date.
Finished To force a rebuild of up-to-date targets, run again with --force
Running .xgettext/helloworld

Building package gettext:merge in .dub/packages/gettext/1.0.7/gettext/merge/
Starting Performing "debug" build using /Library/D/dmd/bin/dmd for x86_64.
Up-to-date colorize 1.0.5: target for configuration [library] is up to date.
Up-to-date gettext:merge 1.0.7: target for configuration [application] is up to date.
Finished To force a rebuild of up-to-date targets, run again with --force
Running .dub/packages/gettext/1.0.7/gettext/merge/gettext_merge --popath=po --backup=none
WARNING: No ".po" files found at "po", nothing to merge

Make sure to supply their path with the "--popath" option.
Building package gettext:po2mo in .dub/packages/gettext/1.0.7/gettext/po2mo/
Starting Performing "debug" build using /Library/D/dmd/bin/dmd for x86_64.
Up-to-date colorize 1.0.5: target for configuration [library] is up to date.
Up-to-date gettext:po2mo 1.0.7: target for configuration [application] is up to date.
Finished To force a rebuild of up-to-date targets, run again with --force
Running .dub/packages/gettext/1.0.7/gettext/po2mo/gettext_po2mo --popath=po --mopath=mo
WARNING: No ".po" files found at "po".

Make sure to supply their path with the "--popath" option.
```

```
```
```
# PO Template for helloworld.
# Copyright © 2023, Bastiaan Veelo
# This file is distributed under the proprietary license.
# Bastiaan Veelo, 2023.
#
#, fuzzy
msgid ""
msgstr ""

#: source/app.d:6
msgid "Edit source/app.d to start your project."
msgstr ""
msgid ""
msgstr ""

#: source/app.d:6 (main)
msgid "Edit source/app.d to start your project."
msgstr ""
msginit -i po/helloworld.pot -o po/nl_NL.po -l nl_NL
msgid ""
msgstr ""
"Project-Id-Version: v1.0.4-17-gdd94820\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2023-08-28T17:38:30.468051Z\n"
"PO-Revision-Date: 2023-08-28 18:39+0100\n"
"Last-Translator: Bastiaan Veelo <Bastiaan@veelo.net>\n"
"Language-Team: Dutch <vertaling@vrijschrift.org>\n"
"Language: nl_NL\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=(n != 1);\n"

#: source/app.d:6(main)
msgid "Edit source/app.d to start your project."
msgstr "Verander source/app.d om je project te starten."
Project-Id-Version: v1.0.4-17-gdd94820
Report-Msgid-Bugs-To:
POT-Creation-Date: 2023-08-28T17:38:30.468051Z
PO-Revision-Date: 2023-08-28 18:39+0100
Last-Translator: Bastiaan Veelo <Bastiaan@veelo.net>
Language-Team: Dutch <vertaling@vrijschrift.org>
Language: nl_NL
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
Plural-Forms: nplurals=2; plural=(n != 1);

# source/app.d:6
msgid "Edit source/app.d to start your project."
msgstr "Verander source/app.d om je project te starten."

### dub build --c=i18n

```bash
bastiaan$ dub build --c=i18n
  Pre-gen Running commands for helloworld
  Starting Performing "debug" build using /Library/D/dmd/bin/dmd for x86_64.
  Up-to-date mofile 0.2.1: target for configuration [library] is up to date.
  Up-to-date gettext 1.0.7: target for configuration [xgettext] is up to date.
  Up-to-date helloworld ~master: target for configuration [gettext] is up to date.
  Finished To force a rebuild of up-to-date targets, run again with --force
  Running .xgettext/helloworld

  Building package gettext:merge in .dub/packages/gettext/1.0.7/gettext/merge/
  Starting Performing "debug" build using /Library/D/dmd/bin/dmd for x86_64.
  Up-to-date colorize 1.0.5: target for configuration [library] is up to date.
  Up-to-date gettext:merge 1.0.7: target for configuration [application] is up to date.
  Finished To force a rebuild of up-to-date targets, run again with --force

  Running .dub/packages/gettext/1.0.7/gettext/merge/gettext_merge --popath=po --backup=none
  msgmerge po/nl_NL.po po/helloworld.pot --update --backup=none
  Building package gettext:po2mo in .dub/packages/gettext/1.0.7/gettext/po2mo/
  Starting Performing "debug" build using /Library/D/dmd/bin/dmd for x86_64.
  Up-to-date colorize 1.0.5: target for configuration [library] is up to date.
  Up-to-date gettext:po2mo 1.0.7: target for configuration [application] is up to date.
  Finished To force a rebuild of up-to-date targets, run again with --force

  Running .dub/packages/gettext/1.0.7/gettext/po2mo/gettext_po2mo --popath=po --mopath=mo
  msgfmt po/nl_NL.po --no-hash -o mo/nl_NL.mo
  Starting Performing "debug" build using /Library/D/dmd/bin/dmd for x86_64.
  Up-to-date mofile 0.2.1: target for configuration [library] is up to date.
  Up-to-date gettext 1.0.7: target for configuration [default] is up to date.
  Up-to-date helloworld ~master: target for configuration [i18n] is up to date.
  Finished To force a rebuild of up-to-date targets, run again with --force

  Copying files for helloworld...
```

```
import std.stdio, gettext;

void main()
{
    mixin(gettext.main);

    selectLanguage;
    writeln(tr!'Edit source/app.d to start your project.');
}

void selectLanguage()
{
    int choice = -1;
    string[] languages = availableLanguages;
    writeln('Please select a language:');
    writeln('[0] default');
    foreach (i, language; languages)
        writeln('[', i + 1, '] ', language.languageCode);
    readf(' %d', &choice);
    if (choice < 1 || choice > languages.length)
        gettext.selectLanguage(null);
    else
        gettext.selectLanguage(languages[choice - 1]);
}
import std.stdio, gettext;

void main()
{
    mixin(gettext.main);
    selectLanguage;
    writeln(tr!"Edit source/app.d to start your project.");
}

void selectLanguage()
{
    int choice = -1;
    string[] languages = availableLanguages;
    writeln("Please select a language:");
    writeln("[0] default");
    foreach (i, language; languages)
        writeln("[", i + 1, "] ", language.languageCode);
    readf(" %d", &choice);
    if (choice < 1 || choice > languages.length)
        gettext.selectLanguage(null);
    else
        gettext.selectLanguage(languages[choice - 1]);
}
int choice = -1;
string[] languages = availableLanguages;
writeln("Please select a language:");
writeln("[0] default");
foreach (i, language; languages)
  writeln("[", i + 1, "] " , language.languageCode);
readf(" %d", &choice);
if (choice < 1 || choice > languages.length)
  gettext.selectLanguage(null);
else
  gettext.selectLanguage(languages[choice - 1]);
I'm counting one apple.

Я рахую %d яблуко.

Translation: Я рахую %d яблуко.
This morning a neat idea occurred to me for a gettext-like system in D that allows automatic and reliable extraction of all translation strings from a program, that doesn't need an external parser to run over the program source code.

Traditionally, gettext requires an external tool to parse the source code and extract translatable strings. In D, however, we can take advantage of (1) passing the format string at compile-time to gettext(), which then allows (2) using static this() to register all format strings at runtime to a central dictionary of format strings, regardless of whether the corresponding gettext() call actually got called at runtime. (3) Wrap that in a version() condition, and you can have the compiler do the string extraction for you without needing an external source code parser.

Here's a proof of concept:

```d
// File: lang.d
version(extractStr) {
  int[string] allStrings;
  void main() {
```
// File: lang.d
version(extractStr) {
    int[string] allStrings;
    void main() {
        import std.algorithm;
        import std.stdio;
        auto s = allStrings.keys;
        s.sort();
        writeln("string[string] dict = \
        \n[/t%s: \"","\n%]");
    }
}

template gettext(string fmt, Args...)
{
    version(extractStr)
    static this() {
        allStrings[fmt]++; 
    }
    string gettext(Args args) {
        import std.format;
        return format(fmt, args);
    }
}
// File: lang.d

version(extractStr) {
    int[string] allStrings;
    void main() {
        import std.algorithm;
        import std.stdio;
        auto s = allStrings.keys;
        s.sort();
        writeln("string[string] dict = [\n        ");
        s.foreach_kv(k, v => writeln("%s: ")),
       .writeln("];");
    }
}

template gettext(string fmt, Args...) {
    version(extractStr)
    static this() {
        allStrings[fmt]++;
    }
    string gettext(Args args) {
        import std.format;
        return format(fmt, args);
    }
}

// File: main.d

import mod1, mod2;

version(extractStr) {} else
void main() {
    auto names = [ "Joe", "Schmoe", "Jane", "Doe" ];
    foreach (i; 0 .. names.length) {
        fun1(names[i]);
        fun2(5 + cast(int)i*10);
    }
}

// File: mod1.d

import std.stdio;
import lang;

void fun1(string name) {
    writeln(gettext!”Hello! My name is %s.”(name));
}

// File: mod2.d

import std.stdio;
import lang;

void fun2(int num) {
    writeln(gettext!”I'm counting %d apples.”(num));
}

void fun3() {
    writeln(gettext!”Never called, but nevertheless registered!”);
}
// File: main.d
import mod1, mod2;

version(extractStr) {}
else
void main()
{
    auto names = ["Joe", "Schmoe", "Jane", "Doe"];
    foreach (i; 0 .. names.length)
    {
        fun1(names[i]);
        fun2(5 + cast(int)i*10);
    }
    writeln("string[string] dict = [
        %s:" % names, 
        %s: "",
    ]");
}

template gettext(string fmt, Args ...)
{
    version(extractStr)
    static this()
    {
        allStrings[fmt]++;
    }
    string gettext(Args args)
    {
        import std.format;
        return format(fmt, args);
    }
}

// File: mod1.d
import std.stdio;
import lang;

void fun1(string name)
{
    writeln(gettext!"Hello! My name is %s."(name));
}

// File: mod2.d
import std.stdio;
import lang;

void fun2(int num)
{
    writeln(gettext!"I'm counting %d apples."(num));
}

void fun3()
{
    writeln(gettext!"Never called, but nevertheless registered!");
}
// File: mod1.d
import std.stdio;
import lang;
void fun1(string name) {
    writeln(gettext!"Hello! My name is %s."(name));
}

// File: mod2.d
import std.stdio;
import lang;
void fun2(int num) {
    writeln(gettext!"I'm counting %d apples."(num));
}
void fun3() {
    writeln(gettext!"Never called, but nevertheless registered!");
}

dmd -i --version=extractStr -run main.d

// File: main.d
import mod1, mod2;
version(extractStr) {
    if (else)
void main() {
    auto names = ["Joe", "Schmoe", "Jane", "Doe"];     
    foreach (i; 0 .. names.length) {
        fun1(names[i]);
        fun2(5 + cast(int)i*10);
    }
    writeln("string[string] dict = 
    %(
    |%|%)");
}

template gettext(string fmt, Args ...) {
version(extractStr)
static this() {
    allStrings[fmt]++;
}
string gettext(Args args) {
    return 
    // File: lang.d
version(extractStr) {
    int[string] allStrings;
    void main() {
        import std.algorithm;
        import std.stdio;
        auto s = allStrings.keys;
        s.sort();
        writeln("string[string] dict = 
        %(
        |%|%)");
    }
    template gettext(string fmt, Args ...) {
version(extractStr)
static this() {
    allStrings[fmt]++;
}
string gettext(Args args) {
    return 
        // File: mod1.d
import std.stdio;
import lang;
void fun1(string name) {
    writeln(gettext!"Hello! My name is %s."(name));
}

https://forum.dlang.org/post/mailman.2526.1585832475.31109.digitalmars-d@puremagic.com
// File: lang.d

version(extractStr) {
    int[string] allStrings;
    void main() {
        import std.algorithm;
        import std.stdio;
        auto s = allStrings.keys.sort();
        writeln("string[string] dict = [\n        \n        \t" "Hello! My name is %s."::"",
        \t"I'm counting %d apples."::"",
        \t"Never called, but nevertheless registered!": "",
        \n        ];")
    }
}

template gettext(string fmt) {
    version(extractStr)
    static this() {
        allStrings[fmt]++;
    }
    string gettext(ArgS args) {
        import std.format;
        return format(fmt, args);
    }
}

// File: mod1.d

import std.stdio;
import lang;

void fun1(string name) {
    writeln(gettext!"Hello! My name is %s."(name));
}

// File: mod2.d

import std.stdio;
import lang;

void fun2(int num) {
    writeln(gettext!"I'm counting %d apples."(num));
}

void fun3() {
    writeln(gettext!"Never called, but nevertheless registered!");
}

// File: main.d

import mod1, mod2;

version(extractStr) {}
else

void main() {
    auto names = [ "Joe", "Schmoe", "Jane", "Doe" ];
    foreach (i; 0 .. names.length) {
        fun1(names[i]);
        fun2(5 + cast(int)i*10);
    }
    writeln("string[string] dict = [\n    \n    \t" "Hello! My name is %s."::"",
    \t"I'm counting %d apples."::"",
    \t"Never called, but nevertheless registered!": "",
    \n    ];")
}

dmd -i -version=extractStr -run main.d
Let me dust off my wand ;)

```d
struct TranslatedString {
    private string _str;
    string get() {
        return curLang.translate(_str);
    }
    alias get this;
}
template gettext(string str) {
    version(extractStrings) {
        shared static this() {
            ++translatableStrings.require(str); // use require here, even though the ++ works without it.
        }
        enum gettext = TranslatedString(str);
    }
}
```

What does this do? It *still* generates the template, but the key difference is that the TranslatedString type is not a template. An enum only exists in the compiler, it’s as if you pasted the resulting code at the call site. So it should not take up any space, maybe 2 words for the string reference. But only one TypeInfo (if that’s even needed, I’m not sure), and a minor CTFE-call for the construction.

It *will* take up space in the symbol table, but that goes away once compilation is done.
Conclusions
Conclusions

• Minimize obfuscation, minimally invasive.
Conclusions

• Minimize obfuscation, minimally invasive.
• Functionally on par with GNU Gettext.
Conclusions

• Minimize obfuscation, minimally invasive.
• Functionally on par with GNU Gettext.
• Low barrier for usage (integrates well in existing Dub project).
Conclusions

- Minimize obfuscation, minimally invasive.
- Functionally on par with GNU Gettext.
- Low barrier for usage (integrates well in existing Dub project).
- Translator friendly.
Conclusions

• Minimize obfuscation, minimally invasive.
• Functionally on par with GNU Gettext.
• Low barrier for usage (integrates well in existing Dub project).
• Translator friendly.

Use it & report any issues!